MARINESCU MĂDĂLINA – AN IV, IS, GR. 4.1

TEMA 2 – DATC

BACKGROUNG JOBS

Background job-urile sunt task-uri care rulează independent de interfața cu utilizatorul. Aceste pot fi executate fără să interacționeze cu utilizatorul, aplicația le pornește și continuă în același timp să preia comenzi.

Background job-urile sunt utile în aplicațiile cu un volum mare de date de procesat și reduc timpul de răspuns, îmbunătățind disponibilitatea.

Criteriul cel mai important în alegerea implementării unui task ca background job este dacă acesta poate rula fără să interacționeze cu utilizatorul și fără ca interfața cu utilizatorul să aștepte ca task-ul să își încheie activitatea.

Tipuri de background job-uri:

- Task-uri care folosesc intens procesorul (de ex.: calcule matematice)
- Operații de intrare-ieșire (de ex.: indexarea fișierelor)
- Batch jobs (de ex.: actualizarea unor date peste noapte)
- Task-uri care consumă mult timp (de ex.: finalizarea unei comenzi)
- Procesarea datelor sensibile (de ex:: transferarea datelor sensibile printr-un Gatekeeper către un background job izolat)

Triggere:

- Event-driven triggers

Background job-urile sunt declanșate de apariția unui eveniment, de exemplu:

- Interfața cu utilizatorul sau alt task plasează un mesaj într-o coadă verificată de către background job. La apariția unui nou mesaj în coadă, acesta îl preia și îl folosește ca dată de intrare.
- Interfața cu utilizatorul sau alt task salvează sau actualizează o valoare într-o zonă de stocare. Background job-ul detectează modificarea, preia data și o procesează.
- Interfața cu utilizatorul sau alt task face o cerere la un endpoint, cum ar fi HTTPS URI, API expus ca web service. Data este pasată ca parte a cererii, iar endpoint-ul sau web service-ul invocă background job-ul care procesează data.

- Schedule-driven triggers

În acest caz se folosește un timer pentru a porni background job-ul, de exemplu:

- Un timer care rulează local în cadrul aplicației sau în cadrul sistemului de operare al aplicației invocă background job-ul la un interval de timp regulat.
- Un timer care rulează în cadrul alte aplicații sau un serviciu de tipul Azure Scheduler trimite o cerere către un API sau către un web service la un interval de timp regulat și acesta invocă background job-ul.
- Un proces separat sau o aplicație pornește un timer care provoacă invocarea background job-ului la un anumit interval de timp.

Hosting environment:

Background job-urile pot fi găzduite pe diverse platforme Azure:

- Azure Web Apps and WebJobs – pentru executarea unor job-uri uzuale în cadrul unor aplicații web
- Azure Virtual Machines – în cazul utilizării unui Windows Service sau Windows Task Scheduler
- Azure Batch – pentru job-uri consumatoare de resurse care rulează pe o colecție de mașini virtuale
- Azure Kubernetes Service – permite un environment pentru Kubernetes în Azure

Rezultate:

Background job-urile se execută asincron într-un proces separat de cel al interfeței cu utilizatorul care le-a invocat. De obicei sunt operații de tipul „fire and forget”, execuția lor neavând impact asupra interfeței cu utilizatorul sau procesului apelant, deci acestea nu pot detecta când background job-ul și-a incheiat execuția. Pentru ca background job-ul să comunice cu interfața cu utilizatorul trebuie implementat un mecanism special.

Exemple:

1. Background job care trimite email-uri:

```
public class SimpleSendEmailJob : BackgroundJob<SimpleSendEmailJobArgs>, ITransientDependency
{
    private readonly IRepository<User, long> _userRepository;
    private readonly IEmailSender _emailSender;

    public SimpleSendEmailJob(IRepository<User, long> userRepository, IEmailSender emailSender)
    {
        _userRepository = userRepository;
        _emailSender = emailSender;
    }

    [UnitOfWork]
    public override void Execute(SimpleSendEmailJobArgs args)
    {
        var senderUser = _userRepository.Get(args.SenderUserId);
        var targetUser = _userRepository.Get(args.TargetUserId);

        _emailSender.Send(senderUser.EmailAddress, targetUser.EmailAddress, args.Subject, args.Body);
    }
}
```

Background job-ul trebuie înregistrat prin dependency injection. Acest lucru se realizează prin implementarea ITransientDependency.

Metoda SimpleSendEmailJob primește ca parametri repository-ul user-ului pentru a prelua mail-urile user-ului și un serviciu IEmailSender care trimite mail-urile.

Metoda Execute trebuie să fie serializabilă și primește ca argument un parametru de tipul unei clase generice. Aceasta este:

```
[Serializable]
public class SimpleSendEmailJobArgs
{
    public long SenderUserId { get; set; }

    public long TargetUserId { get; set; }

    public string Subject { get; set; }

    public string Body { get; set; }
}
```

2. Background job care rulează după ce aplicația web a fost închisă:

Se folosește HostingEnvironment API disponibil în .NET Framework 4.5.2, care permite rularea background job-urilor în aplicații web ASP .NET. Pentru task-uri care durează foarte mult, utilizatorul poate porni task-ul, iar apoi să închidă aplicația în timp ce background job-ul rulează. În exemplul următor, utilizatorul pornește task-ul prin apăsarea unui buton de submit și închide aplicația imediat după. Background job-ul rulează în continuare și scrie numerele de la 1 la 20 într-un fișier .txt cu un delay de 1.5 sec la fiecare scriere.

```
public class Worker
{
    string filePath = @"d:/test.txt";

    1 reference
    public void StartProcessing(CancellationToken cancellationToken = default(CancellationToken))
    {
        try
        {
            if (File.Exists(filePath))
                File.Delete(filePath);

            using (File.Create(filePath)) { }
            using (System.IO.StreamWriter file = new System.IO.StreamWriter(filePath))
            {
                for (int index = 1; index <= 20; index++)
                {
                    //execute when task has been cancel
                    cancellationToken.ThrowIfCancellationRequested();
                    file.WriteLine("Its Line number : " + index + "\n");
                    Thread.Sleep(1500);   // wait to 1.5 sec every time
                }

                if (Directory.Exists(@"d:/done"))
                    Directory.Delete(@"d:/done");
                Directory.CreateDirectory(@"d:/done");
            }
        }
        catch (Exception ex)
        {
            ProcessCancellation();
            File.AppendAllText(filePath, "Error Occured : " + ex.GetType().ToString() + " : " + ex.Message);
        }
    }
    1 reference
    private void ProcessCancellation()
    {
        Thread.Sleep(10000);
        File.AppendAllText(filePath, "Process Cancelled");
    }
}
```