[Hangfire](http://hangfire.io)

(Open-source library) **2013**

It simplifies asynchronous tasks in your application by handling them in the background.

## References:

* [Main Domain](https://www.hangfire.io/extensions.html)
* [Youtube Video](https://www.youtube.com/watch?v=lsP5ZjcFtMM)
* ChatGpt

# Features

**Ease of Use**: Hangfire provides a simple API for defining and scheduling background jobs, requiring minimal configuration.

**Support for Various Job Types**: It allows for fire-and-forget, recurring, delayed, and continuation jobs, catering to diverse scheduling needs.

**Dashboard for Monitoring**: The built-in web dashboard offers real-time job monitoring, logging, and management capabilities.

**Reliability**: Hangfire automatically retries failed jobs and uses persistent storage to track job statuses, ensuring reliability.

**Community Support and Documentation**: With an active community and comprehensive documentation, Hangfire provides valuable resources for developers.

# Step

## Install Below Packages

* Hangfire.AspNetCore(Nuget Package) –v 1.8.14
* Hangfire.SqlServer(Nuget Package) –v 1.8.14
* System.Data.SqlClient(Nuget Package) –v latest

## Write code in Program.CS

### Service of Hangfire

builder.Services.AddHangfire((sp, config)=>{

var configuration = sp.GetService<IConfiguration>(); // or GetRequiredService<IConfiguration>()

var connectionString = configuration["connection string location"];

config.UseSqlServerStorage(connectionString);

});

### Dashboard middleware

app.UseHangfireDashboard();

### Server service

builder.Services.AddHangfireService();

# Note:

1.After running the project, the necessary tables for Hangfire will be automatically created.

2.A dashboard will also be available for monitoring jobs.

# Practice

## 1.Fire and Forgot

This is like **immediately**.

BackgroundJob.Enqueue(() => method calling);

**Executing**

## 2.Delayed

This is like a **specified time interval.**

var JobId=BackgroundJob.Schedule(()=> method calling, TimeSpan);

**Scheduling**

var Status =BackgroundJob.Delete(JobId);

**Deleting**

var Status = BackgroundJob.Reschedule(JobId,TimeSpan);

**Rescheduling**

var Status = BackgroundJob.Requeue(JobId);

**Requeue**

## 3.Recurring

This is like a **CRON schedule, many times.** runs repeatedly on a schedule (e.g., every minute, daily, or weekly)

\* \* \* \* \* \*

| | | | | |

| | | | | +---- Day of the week (0 - 7) (Sunday is 0 or 7)

| | | | +------ Month (1 - 12)

| | | +-------- Day of the month (1 - 31)

| | +---------- Hour (0 - 23)

| +------------ Minute (0 - 59)

+-------------- Second (0 - 59)

**Add & Edit**   
 RecurringJob.AddOrUpdate<class name>(

jobId, // job Id

service => service.”method call”,

"\* \* \* \* \* \*", // cron expression

TimeZoneInfo.Local // timezone

);

**Delete**

RecurringJob.RemoveIfExists(jobId);

## 4.Continuations

**when the parent job has finished.**

// Enqueue the initial background job

string jobId = BackgroundJob.Schedule((method), TimeSpan.FromMinutes(minutes));

// Schedule a continuation job after the initial job completes

BackgroundJob.ContinueJobWith(jobId, method);