Audit Trail Component

Version v1.0

This document describes [Audit Trail Component] features, how to register and how to use it.

Contents

Introduction

Audit trail component stores database actions (Add, Update, Delete), business actions (you can specify them), sends notifications and logs its own errors, also it's dynamic as it gives user the ability to specify [Business actions], [Emails that notifications will be sent to], [Notification's subject] and [Notification's Message], also you can search for all the data has been stored by the component by predefined methods.

Framework: .net standard 2.1

Packages :

- Microsoft.EntityFrameworkCore.SqlServer 3.1.3
- Newtonsoft.Json 12.0.3

Hangfire Packages :

- Hangfire.AspNetCore
- Hangfire.SqlServer

Features :

- Store database actions and business actions.
- Send notifications.
- Log send notification process, and retry if it's failed.
- Log errors.
- Dynamic: User can specify actions and emails and specify if the component should send notification when a specific action happened.

Recommendation :

use background job to execute component methods, to ensure that the original request time will not be increased.

Database Tables

Audit.Action

to store actions lookup data

Audit.ActionUserGroup

to make relation between [action], and emails, so if this action happened then the component will send notification to the related emails with the specified message.

Audit.AuditTrail

to store the audit data

Audit.ErrorLog

to log the component errors

Audit.Notification

to store the notifications that the component will send to emails

Audit.NotificationLog

to log the send notification process, component will store every (send email) retry and store if it's done successfully or not with error message

How to Register

- In your project, you need to install Hangfire nugetpackages [`Hangfire.AspNetCore, Hangfire.SqlServer`] , or any package to perform background processing.

Startup class

- In method [`ConfigureServices`]

```
#region hangfire
    // Add Hangfire services.
    services.AddHangfire(configuration => configuration
        .SetDataCompatibilityLevel(CompatibilityLevel.Version_170)
        .UseSimpleAssemblyNameTypeSerializer()
        .UseRecommendedSerializerSettings()
        .UseSqlServerStorage(Configuration.GetConnectionString("DefaultConnection"), new SqlServerStorageOptions
        {
            CommandBatchMaxTimeout = TimeSpan.FromMinutes(5),
            SlidingInvisibilityTimeout = TimeSpan.FromMinutes(5),
            QueuePollInterval = TimeSpan.Zero,
            UseRecommendedIsolationLevel = true,
            UsePageLocksOnDequeue = true,
            DisableGlobalLocks = true
        }));

    // Add the processing server as IHostedService
    services.AddHangfireServer();

    #endregion

    #region Audit
    services.AddScoped<IEmailService, EmailService>();

    //AddAuditTrail classes
    services.AddAuditTrail(Configuration.GetConnectionString("DefaultConnection"),
         new List<AuditActionDto> {
            new AuditActionDto() { ActionCode ="CreateOrder",SendNotification=true,NotificationFromEmail="admin@gmail.com"
                                  ,NotificationEmails="ahmed@yahoo.com,ali@hotmail.com",NotificationSubject="subject "
                                  ,NotificationMessage="sample message"},
            new AuditActionDto() { ActionCode = "DeleteOrder",SendNotification=true,NotificationFromEmail="admin@gmail.com",
                                  NotificationEmails="ahmed2@yahoo.com,ali2@hotmail.com",NotificationSubject="subject "  ,
                                   NotificationMessage="sample message2" },
            new AuditActionDto() { ActionCode = "UpdateOrder" ,SendNotification=true},
           new AuditActionDto() { ActionCode = "NewUpdateOrder" ,SendNotification=false}});


    //register background job to Check Notifications to send emails
    var sp = services.BuildServiceProvider().CreateScope().ServiceProvider;
    var notificationService = sp.GetService<INotificationPublicService>();
    JobStorage.Current = new SqlServerStorage(Configuration.GetConnectionString("DefaultConnection"),
    new SqlServerStorageOptions
    {
        CommandBatchMaxTimeout = TimeSpan.FromMinutes(5),
        SlidingInvisibilityTimeout = TimeSpan.FromMinutes(5),
        QueuePollInterval = TimeSpan.Zero,
        UseRecommendedIsolationLevel = true,
        UsePageLocksOnDequeue = true,
        DisableGlobalLocks = true
    });

    RecurringJob.AddOrUpdate(() => notificationService.CheckNotificationsAndSend(), Cron.Daily);

    #endregion
```

How to use

- Inject this interface to your class [IAuditTrailService]
- To store business action

```
BackgroundJob.Enqueue(() => AuditTrailService.SaveCustomActionsAuditTrailAsync("DeleteOrder", "sample data by hangfire", "username1"));
```

- To store database action ,use the below code before Entity framework save changes method

```
BackgroundJob.Enqueue(() =>AuditTrailService.SaveDbActionsAuditTrailAsync(ApplicationDbContext.ChangeTracker.Entries()
      .Select(a =>
      new DatabaseChangesDto
      {
          Entity = a.Entity,
          OriginalValues = a.OriginalValues.ToObject(),
          CurrentValues = a.CurrentValues.ToObject()
      }).ToList()));
```

- To search for audit data

```
//Inject this interface [IAuditTrailService]

AuditTrailFilterDto model = new AuditTrailFilterDto {Audit_ActionCode="DeleteOrder" };
var result=await AuditTrailService.Search(model);
```

- To search for Notifications

```
//Inject this interface [INotificationPublicService]
NotificationFilterDto model = new NotificationFilterDto { Audit_ActionCode="DeleteOrder" };
var result=await NotificationPublicService.SearchNotifications(model)
```

- to search for notification log

```
//Inject this interface [IErrorLogPublicService]

NotificationLogFilterDto model = new NotificationLogFilterDto { };
 var result = await NotificationService.SearchNotificationLogs(model);
```

- to search for error logs

```
//Inject this interface [IErrorLogPublicService]

ErrorLogFilterDto model4 = new ErrorLogFilterDto();
var result = await ErrorLogPublicService.Search(model4);
```