```
using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Linq;

using Microsoft.AspNetCore.SignalR;

using Mallenom;

using Viscont.Core.Service.ImageDataTransmission.Hubs;

namespace
Viscont.Core.Service.ImageDataTransmission.Data;

public class ImageRepository : IImageRepository
{
	#region Data

	private readonly ConcurrentDictionary<Guid,
LifeTimeImage> _imageRepository;
	private readonly IHubContext<NotificationHub>
_hubContext;

	#endregion

	#region .ctor

	public
ImageRepository(IHubContext<NotificationHub>
hubContext)
	{
		Verify.Argument.IsNotNull(hubContext,
nameof(hubContext));

		_imageRepository = new
ConcurrentDictionary<Guid, LifeTimeImage>();
		_hubContext = hubContext;
	}

	#endregion

	public ImageEntry GetByGuid(Guid guid)
	{
		if(_imageRepository.TryGetValue(guid, out
var imr))
			return imr.ImageMetadataEntry;
		throw new Exception("Not found file");
	}

	public bool TryGetByGuid(Guid imageId, out
ImageEntry entry)
	{
		if(_imageRepository.TryGetValue(imageId,
out var imr))
		{
			entry = imr.ImageMetadataEntry;
			return true;
		}
		entry = default;
		return false;
	}

	public int GetCount() =>
_imageRepository.Count;

	private async void Notify(string @event, Guid
imageGuid)
	{
		await _hubContext.Clients.All
			.SendAsync(@event, imageGuid)
			.ConfigureAwait(continueOnCapturedContext:
false);
	}

	public void Add(Guid imageId, ImageEntry
imageMetadataRepository)
	{
		var lti = new LifeTimeImage(imageId,
imageMetadataRepository, Remove);
		if (_imageRepository.TryAdd(imageId,
lti))
		{
			lti.Start();
			Notify(@"NewImage", imageId);
		}
	}

	public bool Remove(Guid imageId)
	{
		if(!_imageRepository.TryRemove(imageId,
out var imr)) return false;

		imr.Dispose();

		Notify(@"RemoveImage", imageId);

		return true;
	}
}
```