Appsetting

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*",

"ImgageEffectServicesPlugIn": "Effect1||Effect2||Effect3"

}

public class ImageEntity

{

public string Name { get; set; }

public byte[] Image { get; set; }

public int Pixel { get; set; }

public string [] Effects { get; set; }

}

public class ImageProcessingRequest

{

public List<ImageEntity> Images { get; set; }

}

public class ImageProcessingResponse

{

public List<ImageEntity> Images { get; set; }

}

public interface IImageOperationService

{

public string ServiceType { get; set; }

public Task<ImageEntity> PerformOperationAysnc(ImageEntity imageEntity);

}

public interface IImageOperationServiceCollection

{

public Task<ImageEntity> PerformOperations(ImageEntity imageEntity);

}

public class Effect1OperationService : IImageOperationService

{

public string ServiceType { get; set; }

public Effect1OperationService()

{

ServiceType = "Effect1";

}

public async Task<ImageEntity> PerformOperationAysnc(ImageEntity imageEntity)

{

// Perform operation on image and return the updated image object

imageEntity.Name = imageEntity.Name + "+" + this.ServiceType;

return imageEntity;

}

}

public class Effect2OperationService : IImageOperationService

{

public string ServiceType { get; set; }

public Effect2OperationService()

{

ServiceType = "Effect2";

}

public async Task<ImageEntity> PerformOperationAysnc(ImageEntity imageEntity)

{

// Perform operation on image and return the updated image object

imageEntity.Name = imageEntity.Name + "+" + this.ServiceType;

return imageEntity;

}

}

public async Task<ImageEntity> PerformOperationAysnc(ImageEntity imageEntity)

{

// Perform operation on image and return the updated image object

imageEntity.Name = imageEntity.Name + "+" + this.ServiceType;

return imageEntity;

}

|  |
| --- |
| public class ImageOperationServiceCollection: IImageOperationServiceCollection  {  private readonly IConfiguration \_config;  private IEnumerable<IImageOperationService> \_injectedImageOperationServices;  private IEnumerable<string> \_mgageEffectServicesPlugIn;  public ImageOperationServiceCollection(IConfiguration config, IEnumerable<IImageOperationService> ImageOperationServices)  {  \_config = config;  \_injectedImageOperationServices = ImageOperationServices;  \_mgageEffectServicesPlugIn = GetAppsettingValues("ImgageEffectServicesPlugIn");  }  public async Task<ImageEntity> PerformOperations(ImageEntity imageEntity)  {  if (imageEntity != null && imageEntity.Effects != null && \_mgageEffectServicesPlugIn != null)  {  foreach (var serviceType in imageEntity.Effects)  {  if (\_mgageEffectServicesPlugIn.Contains(serviceType))  {  var service = \_injectedImageOperationServices.Where(x => x.ServiceType == serviceType).FirstOrDefault();  if (service != null)  {  imageEntity = await service.PerformOperationAysnc(imageEntity);  }  }  }  }  return imageEntity;  }  public IEnumerable<string> GetAppsettingValues(string key)  {  var result = \_config.GetValue<string>(key);  return result.ToString().Split("||");  }  } |

|  |
| --- |
| using ImageProcessing;  var builder = WebApplication.CreateBuilder(args);  // Add services to the container.  builder.Services.AddScoped<IImageOperationService, Effect1OperationService>();  builder.Services.AddScoped<IImageOperationService, Effect2OperationService>();  builder.Services.AddScoped<IImageOperationService, Effect3OperationService>();  builder.Services.AddScoped<IImageOperationServiceCollection, ImageOperationServiceCollection>();  builder.Services.AddControllers();  // Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle  builder.Services.AddEndpointsApiExplorer();  builder.Services.AddSwaggerGen();  var app = builder.Build();  // Configure the HTTP request pipeline.  if (app.Environment.IsDevelopment())  {  app.UseSwagger();  app.UseSwaggerUI();  }  app.UseHttpsRedirection();  app.UseAuthorization();  app.MapControllers();  app.Run(); |

|  |
| --- |
| [ApiController]  [Route("[controller]")]  public class ImageProcessorController : ControllerBase  {  private readonly ILogger<ImageProcessorController> \_logger;  private IImageOperationServiceCollection \_imageOperationServiceCollection;  public ImageProcessorController(ILogger<ImageProcessorController> logger, IImageOperationServiceCollection imageOperationServiceCollection)  {  \_logger = logger;  \_imageOperationServiceCollection = imageOperationServiceCollection;  }  /// <summary>  /// This method can be processed multiple Imgages with multiple effects  /// </summary>  /// <param name="imageProcessingRequest"></param>  /// <returns></returns>  [HttpPost]  public async Task<ImageProcessingResponse> ProsessImagesAsync([FromBody] ImageProcessingRequest imageProcessingRequest)  {  List<Task> tasks = new List<Task>();  ImageProcessingResponse respone = new ImageProcessingResponse();  respone.Images = new List<ImageEntity>();  if (imageProcessingRequest != null && imageProcessingRequest.Images != null)  {  foreach (var image in imageProcessingRequest.Images)  {  Task<ImageEntity> result = \_imageOperationServiceCollection.PerformOperations(image);  tasks.Add(result);  }  await Task.WhenAll(tasks);  foreach (var task in tasks)  {  respone.Images.Add(((Task<ImageEntity>)task).Result);  }  }  return respone;  }  } |