namespace ApiConnection.Models;

public class Post

{

public int Id { get; set; }

public int UserId { get; set; }

public string? Title { get; set; }

public string? Body { get; set; }

}

// Interface to abstract the service – this enables testing and decoupling

namespace ApiConnection.Services;

using ApiConnection.Models;

public interface IApiService

{

Task<List<Post>> GetUserListAsync();

}

// Concrete implementation of the API service

namespace ApiConnection.Services;

using ApiConnection.Models;

using System.Net.Http;

using System.Net.Http.Json;

public class ApiService : IApiService

{

private readonly HttpClient \_httpClient;

// HttpClient is injected via dependency injection

public ApiService(HttpClient httpClient)

{

\_httpClient = httpClient;

// Set base address only once. Optional if configured elsewhere.

\_httpClient.BaseAddress = new Uri("https://jsonplaceholder.typicode.com/");

}

// Fetches a list of posts from the API

public async Task<List<Post>> GetUserListAsync()

{

var response = await \_httpClient.GetAsync("posts/");

response.EnsureSuccessStatusCode();

// Returns deserialized list or an empty one if null

return await response.Content.ReadFromJsonAsync<List<Post>>() ?? new List<Post>();

}

}

// Register the service and HttpClient in your MauiProgram.cs or Startup.cs

builder.Services.AddHttpClient<IApiService, ApiService>();

using ApiConnection.Services;

using CommunityToolkit.Mvvm.ComponentModel;

using CommunityToolkit.Mvvm.Input;

public partial class UsersViewModel : ObservableObject

{

private readonly IApiService \_apiService;

public UsersViewModel(IApiService apiService)

{

\_apiService = apiService;

}

// Command to fetch user list from API

[RelayCommand]

private async Task GetListUserAsync()

{

var list = await \_apiService.GetUserListAsync();

if (list != null)

{

foreach (var item in list)

{

Console.WriteLine($"id: {item.Id}, Title: {item.Title}, Body: {item.Body}");

}

}

}

}

dotnet add package Refit

Once the installation of our package is complete, the model that represents a publication will be defined:

namespace ApiConnection.Models

{

public class Post

{

public int Id { get; set; }

public int UserId { get; set; }

public string? Title { set; get; }

public string? Body { set; get; }

}

}

Then an interface is defined that represents the API calls and that Refit will be using.

using ApiConnection.Models;

using Refit;

namespace ApiConnection.Services

{

public interface IApiServices

{

//method and resource where the request will be made

[Get("/posts/")]

Task<List<Post>> GetPostListAsync();

}

}

Next, we are going to create our Refit client in the ApiService service that is responsible for interacting with the API.

// ===============================

// CLASS: ApiService

// Implements logic and delegates to the injected Refit client

// ===============================

using ApiConnection.Models;

namespace ApiConnection.Services;

public class ApiService

{

private readonly IApiServices \_apiService;

// The Refit client (IApiServices) is injected via constructor

public ApiService(IApiServices apiService)

{

\_apiService = apiService;

}

// Fetches the list of posts via the Refit client

public async Task<List<Post>> GetPostsAsync()

{

return await \_apiService.GetPostListAsync();

}

}

// ===============================

// MAUIPROGRAM.CS

// Registers Refit, services, and ViewModel for DI

// ===============================

using ApiConnection.Services;

using Refit;

public static class MauiProgram

{

public static MauiApp CreateMauiApp()

{

var builder = MauiApp.CreateBuilder();

builder

.UseMauiApp<App>()

.ConfigureFonts(fonts =>

{

fonts.AddFont("OpenSans-Regular.ttf", "OpenSansRegular");

});

// Register the Refit interface with a base URI

builder.Services.AddRefitClient<IApiServices>()

.ConfigureHttpClient(c =>

{

c.BaseAddress = new Uri("https://jsonplaceholder.typicode.com");

});

// Register the concrete service and ViewModel

builder.Services.AddTransient<ApiService>();

builder.Services.AddTransient<UsersViewModel>();

return builder.Build();

}

}

To see an example of how to use the ApiService, we can create an async method, where an instance of the ApiService will be created. **ApiServices**, to call the **GetPostsAsync** andget the data.

// ViewModel or code-behind class

using ApiConnection.Services;

public class PostsViewModel

{

private readonly ApiService \_apiService;

// ApiService is injected via constructor (DI)

public PostsViewModel(ApiService apiService)

{

\_apiService = apiService;

}

// Method to fetch posts and handle errors

public async Task GetListUserAsync()

{

try

{

var posts = await \_apiService.GetPostsAsync();

foreach (var post in posts)

{

Console.WriteLine($"ID: {post.Id}, Title: {post.Title}");

}

}

catch (Exception ex)

{

Console.WriteLine($"Error: {ex.Message}");

}

}

}