Day 27: Using HttpClient with Xamarin.Android

Today, we will look at how we can use HttpClient to read data from a remote API and display it in a List (RecyclerView). We are looking at HttpClient in today’s example, but you can use any library like WebClient, RestSharp or any other library of your choice.

Keeping to the theme of showing Star Wars data in this series, today, we will consume the Star Wars API [https://swapi.co/documentation](https://swapi.co/documentation#films).

To use HttpClient in your Xamarin.Android Application, you should first install the HttpClient Nuget Package –

|  |
| --- |
| PM> Install-Package Microsoft.Net.Http |

Although optional, I highly recommend using [Paul Betts’](https://twitter.com/paulcbetts) [ModernHttpClient](https://www.nuget.org/packages/modernhttpclient/), a library that uses platform-specific networking libraries like NSURLSession on iOS and OkHttp on Android with just a single line of code. In short, your Http calls will be faster when using ModernHttpCilent along with HttpClient

|  |
| --- |
| PM> Install-Package modernhttpclient |

We will be consuming the Films end point <https://swapi.co/documentation#films> in the Star Wars API. We will also be doing this in a PCL as a way to show using class libraries in your Xamarin.Android Application. We will be showing the data in a RecyclerView, please refer to the RecyclerView post to get more detail on how RecyclerView works.

First, let’s add a new PCL class library called “StarWars.Api.Repository”. To it, let’s add Movie.cs file. This class will be indicative of the model that gets returned from the Films API call. I used the handy <http://json2csharp.com/> to automatically generate my model class from the end point URL –

|  |
| --- |
| using System.Collections.Generic;  // ReSharper disable InconsistentNaming - Naming for remote api  namespace StarWars.Api.Repository  {  public class Movie  {  public string title { get; set; }  public int episode\_id { get; set; }  public string opening\_crawl { get; set; }  public string director { get; set; }  public string producer { get; set; }  public string release\_date { get; set; }  public List<string> characters { get; set; }  public List<string> planets { get; set; }  public List<string> starships { get; set; }  public List<object> vehicles { get; set; }  public List<string> species { get; set; }  public string created { get; set; }  public string edited { get; set; }  public string url { get; set; }  }  public class Films  {  public int count { get; set; }  public object next { get; set; }  public object previous { get; set; }  public List<Movie> results { get; set; }  }  } |

Gist file link: <https://gist.github.com/vkoppaka/6ded995bea96702219b7>

Next up is our Http reading code, let’s look at how to create an HttpClient using ModernHttpClient –

|  |
| --- |
| private HttpClient GetHttpClient()  {  var httpClient = new HttpClient(new NativeMessageHandler())  {  BaseAddress = new Uri(ServiceEndPoints.StartWarsApiBaseUri)  };  httpClient.DefaultRequestHeaders.Accept.Clear();  httpClient.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));  return httpClient;  } |

Gist file link: <https://gist.github.com/vkoppaka/1216654df0a5fde2e391>

As you can see from the HttpClient constructor, we are passing NativeMessageHandler which is part of the ModernHttpClient Nuget package. We are also letting the HttpClient know about the BaseAddress of the API and are telling that we will only be accepting json type requests.

Let’s take a look at the method that gets all films data –

|  |
| --- |
| public async Task<Films> GetAllFilms()  {  var httpClient = GetHttpClient();  var response = await httpClient.GetAsync(ServiceEndPoints.GetFilmsUri).ConfigureAwait(false);  if (response.IsSuccessStatusCode)  {  var content = response.Content;  string jsonString = await content.ReadAsStringAsync().ConfigureAwait(false);  return JsonConvert.DeserializeObject<Films>(jsonString);  }  return new Films();  } |

Gist file link: <https://gist.github.com/vkoppaka/c9cf3250193b278c27be>

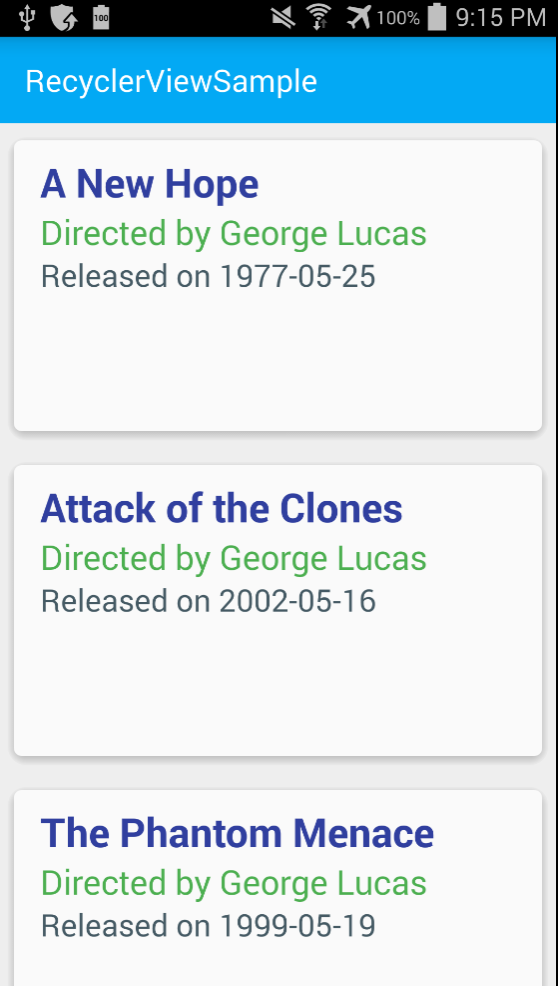
We are using HttpClient’s GetAsync method and reading a Json string of the content. Finally, we are using JSON.NET to deserialize the Json string to the Film class from above.

To show this films data in our RecyclerView we will use the following code –

|  |
| --- |
| using Android.App;  using Android.OS;  using Android.Support.V7.Widget;  using Android.Widget;  using StarWars.Api.Repository;  namespace RecyclerViewSample  {  [Activity(Label = "RecyclerViewSample", MainLauncher = true, Icon = "@drawable/icon")]  public class MainActivity : BaseActivity  {  protected override int LayoutResource  {  get { return Resource.Layout.main; }  }  private RecyclerView recyclerView;  private ProgressBar activityIndicator;  private RecyclerView.LayoutManager layoutManager;  protected override async void OnCreate(Bundle bundle)  {  base.OnCreate(bundle);  recyclerView = FindViewById<RecyclerView>(Resource.Id.recyclerView);  activityIndicator = FindViewById<ProgressBar>(Resource.Id.activityIndicator);  activityIndicator.Visibility = Android.Views.ViewStates.Visible;  layoutManager = new LinearLayoutManager(this, LinearLayoutManager.Vertical, false);  recyclerView.SetLayoutManager(layoutManager);  var repository = new MoviesRepository();  var films = await repository.GetAllFilms();  var moviesAdapter = new MovieAdapter(films.results);  recyclerView.SetAdapter(moviesAdapter);  activityIndicator.Visibility = Android.Views.ViewStates.Gone;  SupportActionBar.SetDisplayHomeAsUpEnabled(false);  SupportActionBar.SetHomeButtonEnabled(false);  }    }  } |

The only change from the previous RecyclerView post to this is that we are using GetAllFilms method from MoviesRepository class and passing it to the adapter.

If you were to run the application, (after rest of the RecyclerView setup is done), it would look like this –



That’s it for today, tomorrow we will look at a similar example using SQLite database.