Este documento explica cómo consumir un api y utilizar la información que esta provee.

**-**Los Apis que utilizaremos son:

1. [https://api.openweathermap.org/data/2.5/weather?q={“Data”}&appid={“Data”}](https://api.openweathermap.org/data/2.5/weather?q=%7b)

El link de la documentación del api:

<https://openweathermap.org/current>

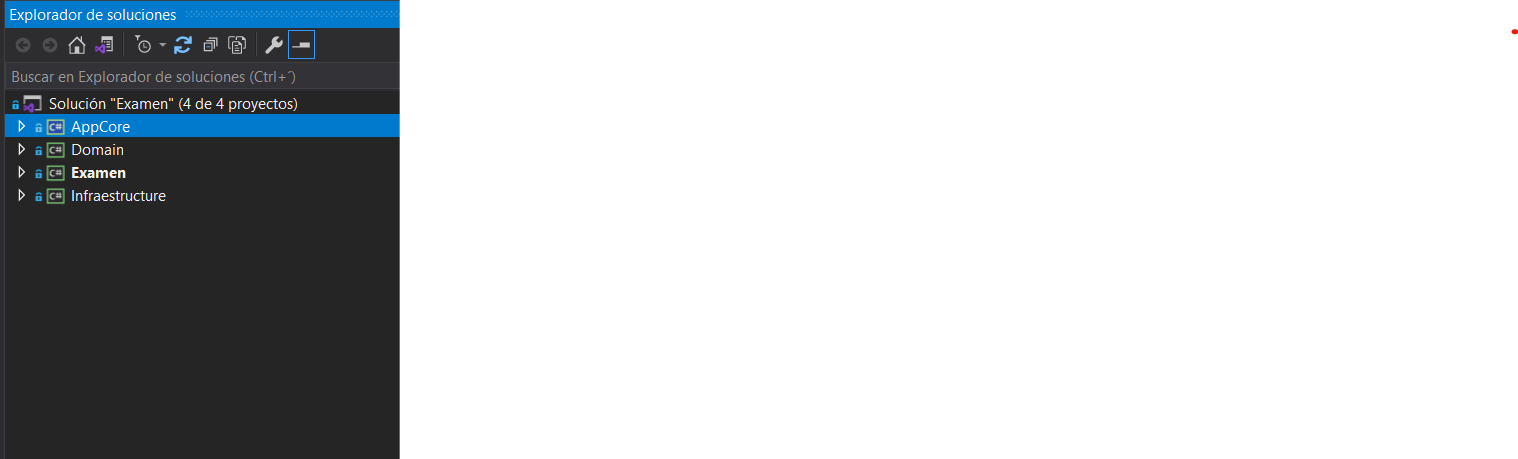
2.[http://api.openweathermap.org/data/2.5/onecall/timemachine?lat={“Data”}&lon={“Data”}&dt={“Data”}&appid={“Data”}](http://api.openweathermap.org/data/2.5/onecall/timemachine?lat=%7b)

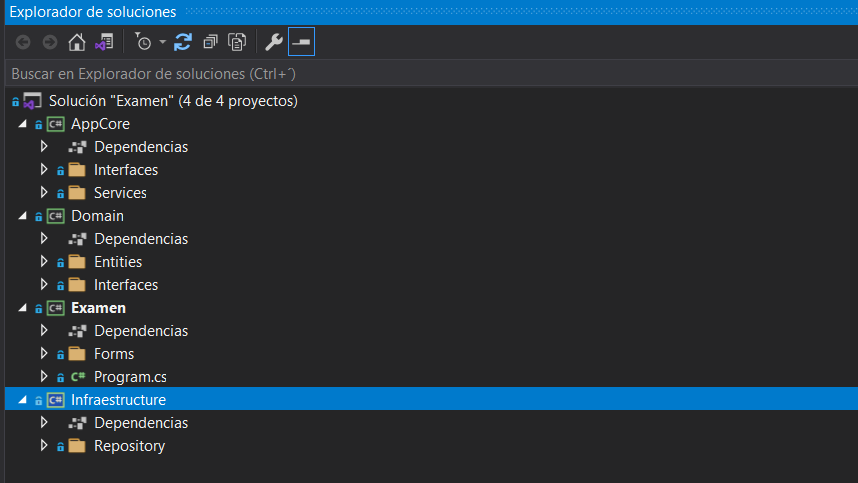
El link de la documentación del api:

<https://openweathermap.org/api/one-call-api#history>

El primer Api lo utilizaremos para obtener la longitud y latitud de una determinada ciudad, estos datos nos servirá para poder utilizar el segundo Api ya que esos datos son necesarios junto a la fecha y el token que tengamos (para este token se debe de suscribir a la página de OpenWeather, funciona si utiliza la suscripción gratis).

Utizaremos el diseño “clea arquitecture” para que el programa pueda ser facilmente expandible en un futuro, por lo tanto, tenemos que crear el proyecto principal el cual solo tendra la presentacion del programa, el proyecto “Domain” para crear la entidades de las apis y otra que llegaremos a utilizar mas adelante, el proyecto “Infraestructure” para tener los procesos que utilizaremos en este proyecto para guardar la informacio que manda el api, y por ultimo el proyecto “AppCore” para crear una capa extra que conecta la presentacion (el proyecto principal) con el proyecto “Infraestructure”:



Una vez creados los proyecto crearemos las siguiente carpetas:

Crearemos en la carpeta “Entities” del proyecto “Domain” las tres siguientes entidades:

1#

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Domain.Entities

{

public class CityInfo

{

public class coord

{

public double lon { get; set; }

public double lat { get; set; }

}

public class weather

{

public string main { get; set; }

public string description { get; set; }

public string icon { get; set; }

}

public class main

{

public double temp { get; set; }

public double pressure { get; set; }

public double humidity { get; set; }

}

public class wind

{

public double speed { get; set; }

}

public class sys

{

public long sunrise { get; set; }

public long sunset { get; set; }

}

public class root

{

public coord coord { get; set; }

public List<weather> weather { get; set; }

public main main { get; set; }

public wind wind { get; set; }

public sys sys { get; set; }

}

}

}

2#

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Domain.Entities

{

public class WeatherInfo

{

public double lat { get; set; }

public double lon { get; set; }

public string timezone { get; set; }

public int timezone\_offset { get; set; }

public Current current { get; set; }

public List<Weather> weather { get; set; }

public class Weather

{

public int id { get; set; }

public string main { get; set; }

public string description { get; set; }

public string icon { get; set; }

}

public class Current

{

public int dt { get; set; }

public int sunrise { get; set; }

public int sunset { get; set; }

public double temp { get; set; }

public double feels\_like { get; set; }

public int pressure { get; set; }

public int humidity { get; set; }

public double dew\_point { get; set; }

public int uvi { get; set; }

public int clouds { get; set; }

public int visibility { get; set; }

public double wind\_speed { get; set; }

public int wind\_deg { get; set; }

public double wind\_gust { get; set; }

public Weather weather { get; set; }

public List<Hourly> hourly { get; set; }

}

public class Snow

{

public double \_1h { get; set; }

}

public class Hourly

{

public int dt { get; set; }

public double temp { get; set; }

public double feels\_like { get; set; }

public int pressure { get; set; }

public int humidity { get; set; }

public double dew\_point { get; set; }

public double uvi { get; set; }

public int clouds { get; set; }

public int visibility { get; set; }

public double wind\_speed { get; set; }

public int wind\_deg { get; set; }

public double wind\_gust { get; set; }

public List<Weather> weather { get; set; }

public Snow snow { get; set; }

}

}

}

3#

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Domain.Entities

{

public class JsonString

{

public int Id { get; set; }

public string json { get; set; }

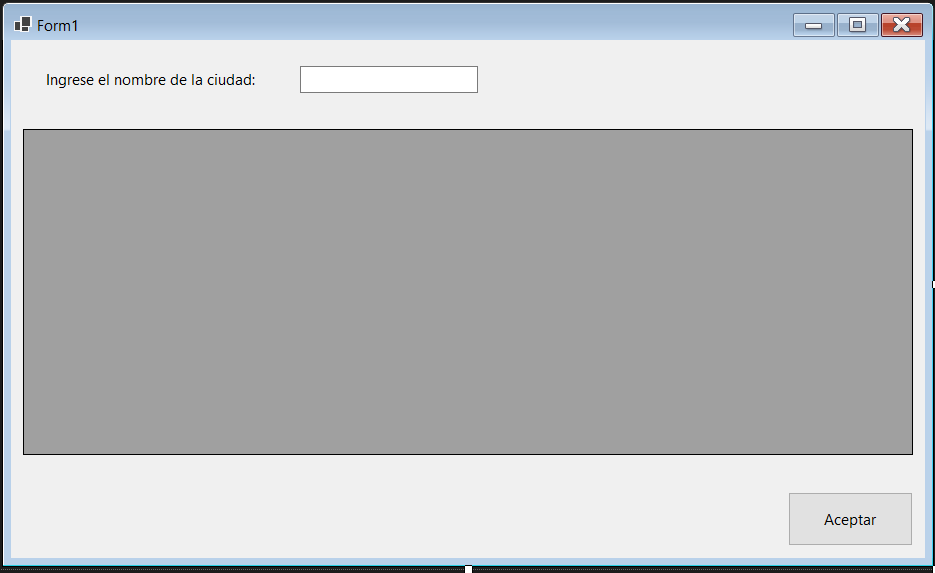
}

}

Seguro te preguntaras: ¿Por qué creamos estas entidades?

Las entidades “1#” y “2#” son para los Apis que consumiremos, estos necesitan que creemos unas variables y clases adecuadas para poder almacenar los datos correctamente (Para saber qué tipo de variables son los datos que devuelven el Api puedes ir a ver la documentación de estos Apis). Y la entidad “3#” tiene como nombre “JsonString” ya que es la que utilizaremos para almacenar en formato Json como un String los datos que los Apis nos devuelve.

Luego tendremos que ir al proyecto principal y crear un formulario de Windows form añadiendo los siguientes elementos: un datagriedview para mostrar los datos de una forma más fácil, un botón para realizar la opción de buscar y consumir los Api, un textbox para que el usuario introduzca el nombre de la ciudad la cual desea buscar, y un label para darle a saber al usuario el dato que estamos pidiendo en el textbox. Quedaría de la siguiente forma:



Ahora crearemos las interfaces en el proyecto “AppCore”(muy importante ponerle la letra I al inicio del nombre de la interface y al final ponerle Services pera tener en claro lo que hace la interface).

1#

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AppCore.interfaces

{

public interface Iservices<T>

{

void Add(T t);

void Delete(T t);

List<T> Read();

int Update(T t);

}

}

2#

using AppCore.interfaces;

using Domain.Entities;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AppCore.Interfaces

{

public interface IJsonServices : Iservices<JsonString>

{

List<WeatherInfo.Current> Convertir();

}

}

Como podemos observar en la primera interface escribimos que recibe un parámetro genérico “T”, esto es para cuando creemos otras entidades y queramos ocupar los métodos que contenga la interface “IServices” simplemente la heredaremos con el parámetro que deseemos similar a como hicimos en la interface número 2.

La interface “2#” se crea para dar servicio (utilizar los métodos en el formulario) a la entidad JsonString ya que así podremos almacenar el json y deserealizarlo desde el proyecto “Infreetructure”.

Una vez tenemos estas interfaces procedemos a crear la clase “BaseServices” en la carpeta Services ya que esta será de tipo genérico y heredará “IServices” para la fácil implementación de nuevas entidades en el proyecto:

using AppCore.interfaces;

using Domain.Interfaces;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AppCore.Services

{

public class BaseServices<T> : Iservices<T>

{

private IModel<T> model;

protected BaseServices(IModel<T> model)

{

this.model = model;

}

public void Add(T t)

{

model.Add(t);

}

public void Delete(T t)

{

model.Delete(t);

}

public List<T> Read()

{

return model.Read();

}

public int Update(T t)

{

return model.Update(t);

}

}

}

Upps!

Podrás observar que hay algo que no hemos creado, y esto es la interface de tipo genérico “IModel”. Esta se crea en la carpeta de interfaces del proyecto “Domain”, contiene los mismos métodos que la interface “IServices” de igual manera, creamos una interface llamada “IJsonModel” que tiene los mismos métodos que “IJsonServices” y hereda “IModel”. Esto se hace para crear la capa extra que les comenté con anterioridad es prácticamente para que el proyecto principal (la presentación) no conecte directamente con los métodos del proyecto “Infraestructure” con la finalidad de que nuestra aplicación sea más segura al ponerse en las manos de otros programadores que introduzcan código en él.

Una vez tenemos estas nueva interface seguiremos creando una clase llamada “JsonServices” en la carpeta “Services” del proyecto “AppCore”

using AppCore.Interfaces;

using Domain.Entities;

using Domain.Interfaces;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AppCore.Services

{

public class JsonServices : BaseServices<JsonString>, IJsonServices

{

IJsonModel jsonModel;

public JsonServices(IJsonModel model):base(model)

{

this.jsonModel = model;

}

public List<WeatherInfo.Current> Convertir()

{

return jsonModel.Convertir();

}

}

}

Y ya tendríamos el proyecto “AppCore” creado con todo lo que necesitamos hasta ahora crearemos lo que necesitamos en el proyecto “Infraestructure”.

Crearemos una clase genérica llamada “BaseRepository” esta heredará la interface “IModel” será heredara por las clases repositorys de las entidades que utilizaremos en este caso la entidad “JsonString”. Pero antes necesitaremos las clases “BinaryReaderExtension” y “RafContext”, esta ultima clase es de tipo genérica creada para almacenar en un archivo de forma binaria con las clases StreamReader y StreamWriter (no se explicará mas detalle de todos los procesos y todo lo que tenga que ver con la clase “RafContext” ya que ese no es el objetivo del tutorial).

Una vez creadas esas 2 clases para proseguimos a crear la clase antes mencionada:

using Domain.Interfaces;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Infraestructure.Repository

{

public abstract class BaseRepositoy<T> : IModel<T>

{

public List<T> data;

protected BaseRepositoy()

{

data = new List<T>();

}

public void Add(T t)

{

data.Add(t);

}

public void Delete(T t)

{

data.Remove(t);

}

public List<T> Find()

{

throw new NotImplementedException();

}

public List<T> Read()

{

return data;

}

public int Update(T t)

{

throw new NotImplementedException();

}

}

}

Crearemos la clase “BinaryJsonRepository”, esta heredará la clase anterior:

using Domain.Entities;

using Domain.Interfaces;

using Newtonsoft.Json;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Infraestructure.Repository

{

public class BinaryJsonRepository : IJsonModel

{

private RAFContext context;

// SIZE es el tamaño que queremos en bits que queremos que deje de espacio por cada almacenamiento del objeto

private const int SIZE = 10000;

public BinaryJsonRepository()

{

context = new RAFContext("json", SIZE);

}

public void Add(JsonString t)

{

context.Create<JsonString>(t);

}

public void Delete(JsonString t)

{

context.Delete(t.Id);

}

public List<JsonString> Read()

{

return context.GetAll<JsonString>();

}

public int Update(JsonString t)

{

throw new NotImplementedException();

}

public List<WeatherInfo.Current> Convertir()

//Este método devuelve una lista de Currents

{

List<WeatherInfo.Current> weather=new List<WeatherInfo.Current>();

foreach (JsonString json in Read())

{

WeatherInfo.Current Info= JsonConvert.DeserializeObject<WeatherInfo.Current>(json.json);

weather.Add(Info);

}

return weather;

}

}

}

Con esta clase creada ya tendremos que poder almacenar el archivo en forma de Json y poder deserealizarlo con el método “Convertir()” .

Ahora iremos al proyecto principal y tendremos que tener escrito en el código del formulario lo siguiente:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Net;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using AppCore.interfaces;

using AppCore.Interfaces;

using Domain.Entities;

using Newtonsoft.Json;

namespace Examen.Forms

{

public partial class Form1 : Form

{

IJsonServices jsonServices;

public Form1(IJsonServices jsonServices)

{

this.jsonServices = jsonServices;

InitializeComponent();

}

// la variable apiKey es el token que recibes despues de registrarte en la pagina de los Apis que estamos utilizando, es necesario para hacer la solicitud de la información.

string apiKey = "cce6c84d83b1a7c75e8a6331e023a33a";

void GetClimate()

{

try

{

using (WebClient web = new WebClient())

{

//La variable date obtiene la fecha actual y la convierte en formato long

long date = ((DateTimeOffset)DateTime.Now).ToUnixTimeSeconds();

//Api numero 2.

string url = string.Format("http://api.openweathermap.org/data/2.5/onecall/timemachine?lat={0}&lon={1}&dt={2}&appid={3}", label2.Tag, label1.Tag, date, "cce6c84d83b1a7c75e8a6331e023a33a");

//Aqui obtenemos la cadena Json al descargar la url que tenemos con los datos

string json = web.DownloadString(url);

JsonString json1 = new JsonString

{

json=json

};

//Almacenamos el Json que obtenemos

jsonServices.Add(json1);

WeatherInfo.Current Info = JsonConvert.DeserializeObject<WeatherInfo.Current>(json);

//Le pasamos todos los horarios que obtenemos del Api al datagriedview que creamos.

dataGridView1.DataSource = GetHourlies();

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

//Este metodo obtiene la longitud y latitud de la ciudad que le pasemos

void GetCity()

{

try

{

using (WebClient web = new WebClient())

{

//Api numero 1.

string url = string.Format("https://api.openweathermap.org/data/2.5/weather?q={0}&appid={1}", txtCity.Text, apiKey);

var json = web.DownloadString(url);

CityInfo.root Info = JsonConvert.DeserializeObject<CityInfo.root>(json);

label1.Tag = Info.coord.lon;

label2.Tag = Info.coord.lat;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.Message);

}

}

//En este metodo llamamos al metodo Convertir() para que nos devuelva una lista de currents los cuales a su vez tiene una lista de Hourlier la cual es la que nos interesa.

public List<WeatherInfo.Hourly> GetHourlies()

{

List<WeatherInfo.Hourly> hourlies=new List<WeatherInfo.Hourly>();

foreach (WeatherInfo.Current a in jsonServices.Convertir())

{

foreach(WeatherInfo.Hourly b in a.hourly)

{

hourlies.Add(b);

}

}

return hourlies;

}

private void Form1\_Load(object sender, EventArgs e)

{

dataGridView1.DataSource = GetHourlies();

}

private void btnOk\_Click(object sender, EventArgs e)

{

GetCity();

GetClimate();

}

}

}

Una vez teniendo el código del formulario ya terminado seguimos implementando la inyección de dependencia en la clase que se crea automáticamente llamada “Program”:

static void Main()

{

Application.SetHighDpiMode(HighDpiMode.SystemAware);

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

var builder = new ContainerBuilder();

builder.RegisterType<BinaryJsonRepository>().As<IJsonModel>();

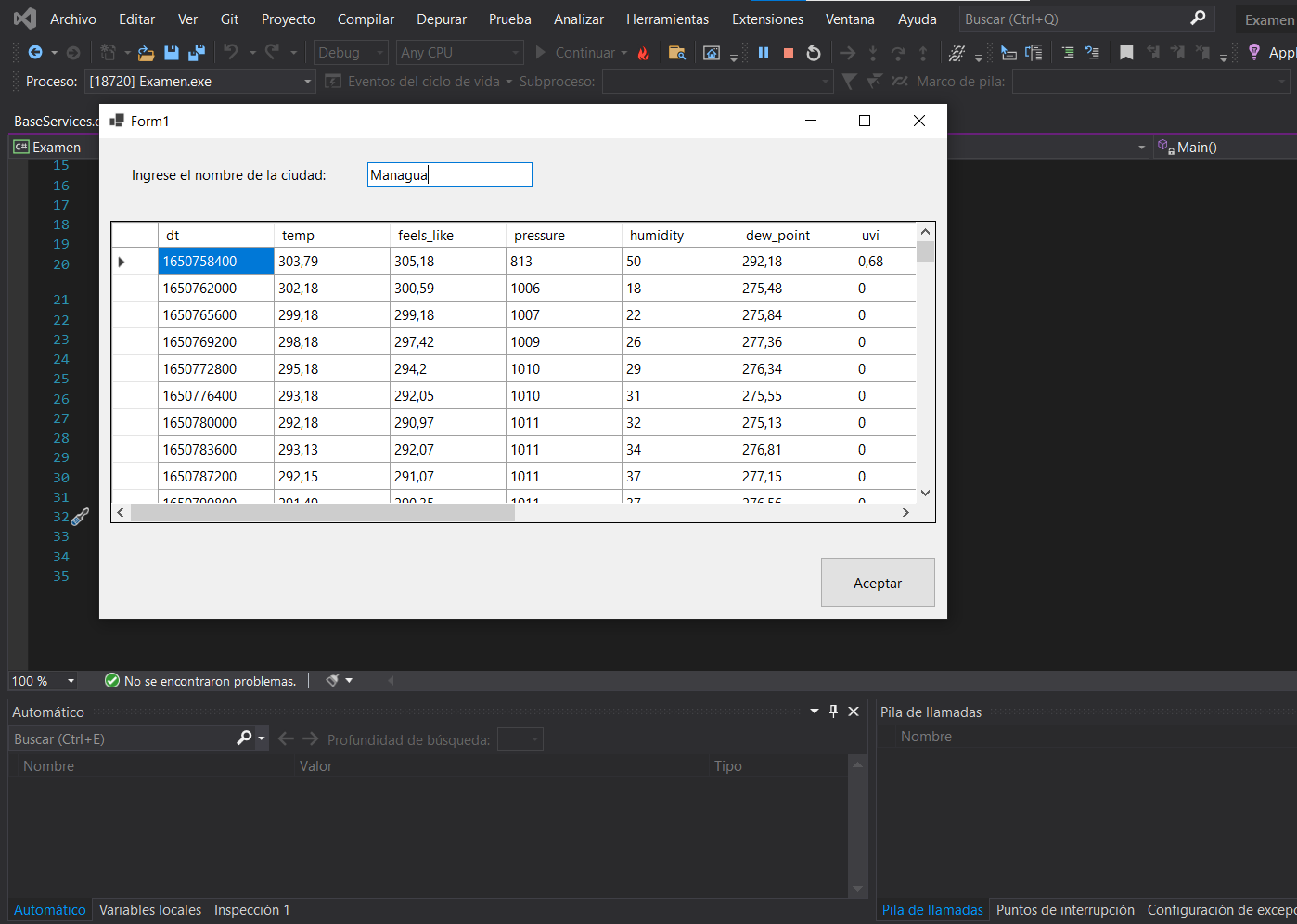
builder.RegisterType<JsonServices>().As<IJsonServices>();

var container = builder.Build();

Application.Run(new Form1(container.Resolve<IJsonServices>()));

}

Con todo esto realizado el programa ya debe de funcionar correctamente de la siguiente manera:



Espero les haiga funcionado el programa y sobre todo les haiga gustado este tutorial.

Los estimo mucho, adiós!! ;)

El proyecto que se muestra en este tutorial es:

https://github.com/rony-obando/Tutorial.git