


```
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using Newtonsoft.Json;
5 using Newtonsoft.Json.Linq;
6 using RestSharp;
7
8 /// <summary>
9 ///
10 /// 0 Código deve retornar
11 ///     Problema resolvido
12 ///
13 /// Não deve ser adicionado nenhum RETURN a mais
14 ///
15 /// </summary>
16
17 class Program
18 {
19     //Nao mexer aqui
20     #region Nao mexer aqui
21     static void Main(string[] args)
22     {
23         Console.WriteLine(Problema());
24         Console.ReadLine();
25     }
26     #endregion
27     //
28     public static string Problema()
29     {
30         try
31         {
32             var ids = new List<string>() { "MLB832035381", "MLB938457671",
33                 "MLB691669454", "MLB837523349" };
34             var dados = BuscarInformacoesML(ids);
35             if (dados != null && dados.Count() == 4)
36                 return "Problema resolvido";
37             else
38                 return "Problema com falha";
39         }
40         catch (Exception ex)
41         {
42             return "Problema com falha";
43         }
44     }
45     /// <summary>
46     ///
47     /// Usando RestSharp e Newtonsoft.Json, implemente um método que
48     /// consulte a api do mercado livre e retorne em um array de "MLItem"
49     /// corrigindo qualquer erro que possa ocorrer
50     ///
51     /// ex https://api.mercadolivre.com/items/MLB832035381
52     ///
53     /// o metodo deverá retornar os seguintes items:
54     ///
55     /// MLB832035381
```

```
55     ///     MLB938457671
56     ///     MLB691669454
57     ///     MLB837523349
58     ///
59     /// </summary>
60     /// <returns></returns>
61
62     private static IEnumerable<MlItem> BuscarInformacoesML(List<string> _ids)
63     {
64         try
65         {
66             // Lista de objetos MlItem.
67             List<MlItem> _retorno = new List<MlItem>();
68             // RestClient com o endpoint
69             var client = new RestClient("https://api.mercadolivre.com");
70
71             // Para cada ID faz uma consulta e adiciona o objeto resultado na
72             // lista.
73             _ids.ForEach(_id =>
74             {
75                 // Configura a requisição com ID e formato.
76                 var request = new RestRequest($"items/{_id}", Method.GET)
77                 { RequestFormat = DataFormat.Json };
78                 // Realiza a requisição.
79                 IRestResponse<List<MlItem>> queryResult =
80                     client.Execute<List<MlItem>>(request);
81                 if (queryResult.StatusCode == System.Net.HttpStatusCode.OK)
82                 {
83                     // Faz a requisição, manda deserializar e adiciona o
84                     // resultado na lista de objetos MlItem.
85                     //_retorno.Add(new JsonSerializer().Deserialize<MlItem>
86                     (queryResult));
87                     _retorno.Add(JsonConvert.DeserializeObject<MlItem>
88                     (queryResult.Content, new JsonSerializer[] { new
89                     UTCDateTimeConverter(), new IntConverter()}));
90                 }
91             });
92             return _retorno;
93         }
94         catch (Exception ex)
95         {
96             return null;
97         }
98     }
99
100     public class MlItem
101     {
102         public string id { get; set; }
103         public string title { get; set; }
104
105         /// <summary>
106         /// Custom Serializer para arredondar o preço e converter para int.
107         /// Preferi manter o tipo de dado como int conforme me que me foi
108         /// solicitado.
109         /// O ideal seria mudar para flot para ter os centavos.
```

```
103     /// </summary>
104     [JsonProperty(ItemConverterType = typeof(IntConverter))]
105     public int price { get; set; }
106
107     /// <summary>
108     /// Tratativa para quando vem null.
109     /// </summary>
110     [JsonProperty(NullValueHandling = NullValueHandling.Ignore)]
111     public int official_store_id { get; set; }
112
113     /// <summary>
114     /// Não teve como deixar essa propriedade com o tipo int, pois a data em
115     ticks precisa de valor maior que o máximo do int. Por isso coloquei
116     como long.
117     /// Poderia usar o tipo DateTime, até porque o dado vem como DateTime com
118     TimeZone, porém como veio o projeto com essa propriedade como int, eu
119     quis manter o mais original possível.
120     /// </summary>
121     [JsonProperty(ItemConverterType = typeof(UTCDateTimeConverter))]
122     private long last_updated { get; set; }
123 }
124
125 public class UTCDateTimeConverter : Newtonsoft.Json.JsonConverter
126 {
127     public override bool CanConvert(Type objectType)
128     {
129         return objectType == typeof(long);
130     }
131
132     public override object ReadJson(JsonReader reader, Type objectType, object
133     existingValue, JsonSerializer serializer)
134     {
135         var _retorno = DateTime.Parse(reader.Value.ToString());
136         return _retorno.Ticks;
137     }
138
139     public override void WriteJson(JsonWriter writer, object value,
140     JsonSerializer serializer)
141     {
142         throw new NotImplementedException();
143     }
144 }
145
146 public class IntConverter : Newtonsoft.Json.JsonConverter
147 {
148     public override bool CanConvert(Type objectType)
149     {
150         return objectType == typeof(int);
151     }
152
153     public override object ReadJson(JsonReader reader, Type objectType, object
154     existingValue, JsonSerializer serializer)
155     {
156         int retorno = 0;
157         if (reader.Path.Equals("price"))
158         {
159             retorno = (int)Math.Round(Double.Parse(reader.Value.ToString()));
160         }
161     }
162
163     public override void WriteJson(JsonWriter writer, object value,
164     JsonSerializer serializer)
165     {
166         throw new NotImplementedException();
167     }
168 }
```

```
152
153         return retorno;
154     }
155     if (reader.Value == null) return null;
156     if (reader.TokenType == JsonToken.Null) return null;
157     if (objectType != typeof(int)) return null;
158
159     int.TryParse(reader.Value.ToString(), out retorno);
160     return retorno;
161 }
162
163 public override void WriteJson(JsonWriter writer, object value, 
164     JsonSerializer serializer)
165 {
166     throw new NotImplementedException();
167 }
```