PASO A PASO 11:
Networking 2 (con Parámetros)

De igual forma que en el caso anterior, vamos a crear una aplicación que muestre en una tabla información de distintos productos. Sin embargo, vamos a modificar la forma en que se obtiene la información.
En el caso anterior, estábamos leyendo un JSON a partir de un servicio de hosting [1]; se trataba del JSON que nos devuelve MercadoLibre para una búsqueda, pero con una estructura más simplificada.
Ahora realmente vamos a utilizar la API de MercadoLibre, mapeando la información tal cual como la recibimos. Además, vamos a pasarle como parámetro la información que queremos obtener: el JSON del caso anterior sólamente listaba productos para la query "dvd". Ahora queremos que el usuario pueda elegir qué término buscar. [2]

[1] https://api.myjson.com/bins/1411cr
[2] *https://api.mercadolibre.com/sites/MLA/search?q=**<palabraClave>***

1. La particularidad de este proyecto respecto al anterior radica en el código de la capa DAO. Las demás capas son prácticamente iguales, y se comportarán igual. La principal diferencia será que cuando el ViewController invoca el método `getProducts` del Service le va a pasar por parámetro el término que el usuario está buscando ("auto", "iphone", "departamento en Belgrano"). Asimismo, cuando el Service invoca el método `getProductsFromAPI` del DAO también le va a pasar ese parámetro extra.

2. Implementación del DAO [***anexo***]:
   a. Por lo tanto, el método que definimos en el DAO deberá esperar un parámetro con el término que se va a buscar, además del **closure**.

```
func getProductsFromAPI(searchTerm: String, daoCompleted:
                        @escaping (([Product]) -> Void)) -> Void
```

   b. Para poder enviar parámetros en un request, debemos crear un diccionario. Este diccionario tomará como claves los nombres de los campos que espera la API, y como valores la información que queremos enviar. De este modo, si queremos enviar un parámetro de nombre "q" con el término de búsqueda, agregaríamos el siguiente código:

```
let queryParams: [String: String] = [:]
queryParams["q"] = searchTerm  // the function parameter
```

c. FInalmente, invocamos el request de igual forma que en el ejercicio anterior; pero además de indicar una url, debemos indicar los parámetros a enviar. Para ello, el método "`request`" de Alamofire permite pasar un parámetro extra llamado "`parameters`" donde pasamos como valor el diccionario creado en el punto anterior.
Cabe destacar que debemos usar la URL de búsqueda hasta el símbolo "?", sin incluirlo.

```
let queryParams: [String: String] = [:]
queryParams["q"] = searchTerm
Alamofire
   .request("https://api.mercadolibre.com/sites/MLA/search",
            parameters: queryParams)
   .responseJSON {
        (response) in
        // instructions go here
}
```

d. Por último, debemos modificar la forma en que parseamos la información recibida. En este caso, el JSON que vamos a recibir es un diccionario (para ver esto podemos cargar la url en nuestro navegador y analizar el resultado: si empieza con '{' es un diccionario, si empieza con un '[' es un array). Dentro de ese diccionario raíz, hay un elemento, con clave 'results', que contiene el array de productos que nosotros queremos utilizar. Luego se itera ese array, y para cada diccionario se crea el Objeto Producto correspondiente.

```
if let rootDictionary = response.value as? [String:AnyObject],
   let resultsArray = rootDictionary["results"] as? [[String:AnyObject]] {

      var productsResultArray: [Product] = []
      for aDictionary in resultsArray {
          if let aProductObject = Product(dictionary: aDictionary) {
              productsResultArray.append(productsResultArray)
          }
      }
}
```

Anexo - DAO

```
import Foundation
import Alamofire

class ProductDAO {

   func getProductsFromAPI(query: String,
                           daoCompleted: @escaping (([Product]) -> Void)) -> Void {
```

```
        var params: [String: String] = [:]
        params["q"] = query

        let url = "https://api.mercadolibre.com/sites/MLA/search"


        Alamofire.request(url, parameters: params).responseJSON {
            (response) in
            // Desde ACA comienza la definicion de un bloque asincronico.
            // En este caso particular, es un bloque que recibe un parametro
            // llamado "response".
            // El parametro recibido va a contener la respuesta del servidor.

            // 'response.value' contiene el JSON leido del servidor,
            // pero por defecto es un AnyObject,
            // por eso es que hay que hacer el 'if let ... as? [String:AnyObject]'

            if let rootDictionary = response.value as? [String:AnyObject] {
                // ACA 'rootDictionary' es el JSON, entendido como un Diccionario.
                // Pero nosotros necesitamos el array de productos,
                // que está en la clave "results" del diccionario.

                if let rootArray = rootDictionary["results"] as? [[String:AnyObject]] {
                    // ACA 'rootArray' es el JSON, entendido como un Array de
                    // Diccionarios; y cada diccionario representa un objeto Product.

                    // Se define el array que va a agrupar todos los objetos parseados.
                    var productsResultArray: [Product] = []

                    // Se itera el array, y se procesa cada diccionario individualmente.
                    for aDictionary in rootArray {
                          // ACA 'aDictionary' es un diccionario
                          // que representa a un objeto Product.

                          let aProductObject = Product(dictionary: aDictionary)
                          productsResultArray.append(aProductObject)

                    }

                    // ACA termine de procesar todos los diccionarios del JSON.
                    // Como ya tengo parseados todos los objetos,
                    // puedo invocar el closure para avisar que se termino el
                    // procesamiento del DAO, y paso como parametro el resultado.
                    daoCompleted(productsResultArray)
                }

                // ACA termina la definicion del bloque asincronico.
            }
        }
    }
}
```