71. Usar HttpClient para consumir servicios REST

Web Services

- La mayoría de las veces, las aplicaciones móviles necesitan acceder y utilizar datos externos, más comúnmente como servicios web basados en REST o SOAP.

- Las aplicaciones de .NET MAUI tienen soporte completo para ambos estilos y el código que crea para interactuar con sus servicios a menudo se puede compartir.

REST

«REST»: ***RE****presentational* ***S****tate* ***T****ransfer*, es un tipo de **arquitectura de desarrollo web** que se apoya totalmente en el estándar **HTTP**. Fue definida en el 2000 por Roy Fielding, uno de los padres de la especificación HTTP y un referente en la arquitectura de redes.

Hoy por hoy, la mayoría de las aplicaciones que se desarrollan para servicios profesionales disponen de una **API REST para el intercambio de información entre el front y el back**. Lo que la hace tan potente es precisamente el aislamiento que proporciona entre la lógica del back-end y cualquier cliente consumidor de éste. Esto le permite ser usada por cualquier tipo de cliente: web, móvil, etc. Así, cualquier dispositivo/cliente que entienda de HTTP puede hacer uso de su propia **API REST** de manera muy simple. Esto ha hecho que en los últimos años este tipo de arquitectura haya ganado peso frente a otras más complejas como **SOAP**, para el intercambio y manipulación de datos.

Operaciones de servicio web

Las solicitudes REST se realizan a través de HTTPS con los mismos verbos HTTP que los exploradores web usan para recuperar páginas web y enviar datos a servidores. Los verbos son:

- **GET**: esta operación se usa para recuperar datos del servicio web.
- **POST**: esta operación se usa para crear un nuevo elemento de datos en el servicio web.
- **PUT**: esta operación se usa para actualizar un elemento de datos en el servicio web.
- **PATCH**: esta operación se usa para actualizar un elemento de datos en el servicio web describiendo un conjunto de instrucciones sobre cómo se debe modificar el elemento.
- **DELETE**: esta operación se usa para eliminar un elemento de datos en el servicio web.

Operaciones de servicio web

Los servicio web que cumplen con REST se definen mediante:

- Identificador URI base.
- Métodos HTTP, como GET, POST, PUT, PATCH o DELETE.
- Tipo de medio para los datos, como notación de objetos JavaScript (JSON).

Operación	HTTP method	URI relativo	Parámetros
Obtener una lista de elementos de tareas pendientes	GET	/api/todoitems/	
Crear un nuevo elemento de tareas pendientes	POST	/api/todoitems/	Un objeto TodoItem con formato JSON
Actualizar un elemento de tareas pendientes	PUT	/api/todoitems/	Un objeto TodoItem con formato JSON
Eliminar un elemento de tareas pendientes	DELETE	/api/todoitems/{id}	

Operaciones de servicio web

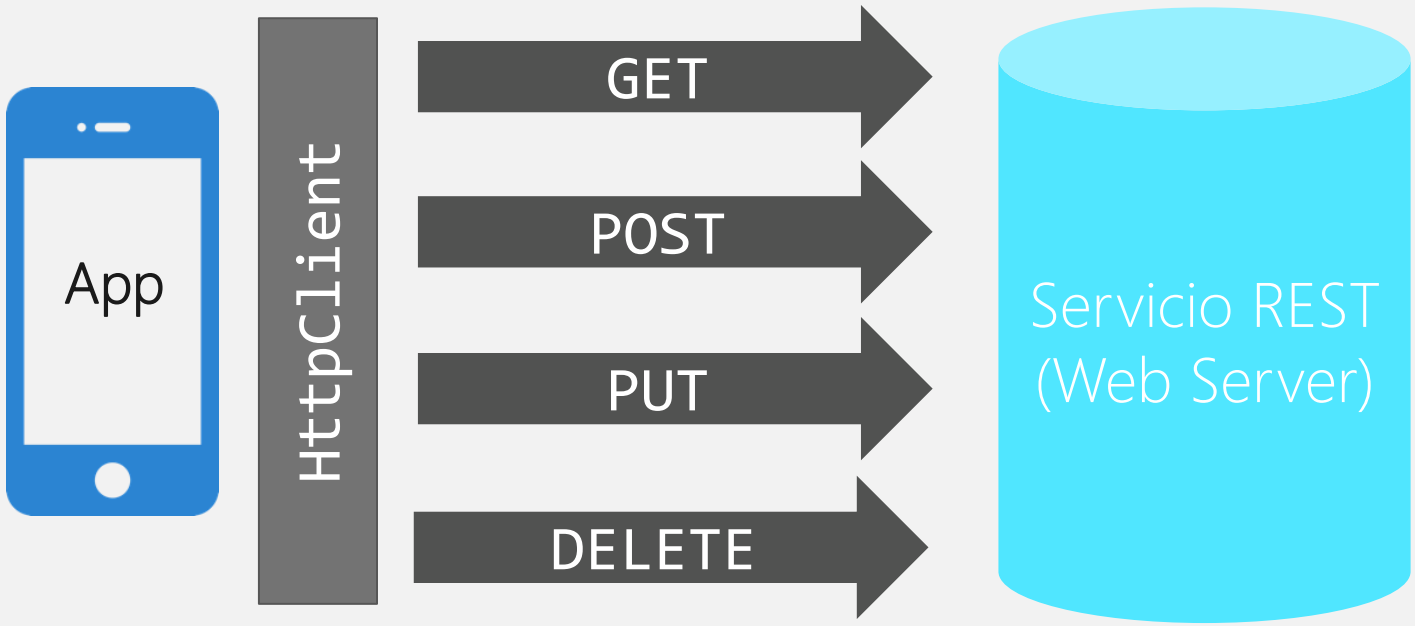
Nos centramos en nuestra aplicación ToDo, la aplicación .NET MAUI y el servicio web usan la clase **TodoItem** para gestionar los datos que se muestran y se envían al servicio web para el almacenamiento:

```
public class TodoItem
{
    public string ID { get; set; }
    public string Name { get; set; }
    public string Notes { get; set; }
    public bool Done { get; set; }
}
```

La propiedad ID se usa para identificar de forma única cada TodoItem objeto y el servicio web usa para identificar los datos que se van a actualizar o eliminar. Por ejemplo, para eliminar un TodoItem con identificador 6bb8a868-dba1-4f1a-93b7-24ebce87e243, la aplicación .NET MAUI envía una solicitud DELETE a https://hostname/api/todoitems/6bb8a868-dba1-4f1a-93b7-24ebce87e243.

Introducción a HttpClient

Las aplicaciones móviles pueden usar la clase **HttpClient** para enviar solicitudes básicas y recibir respuestas a través de HTTP



Creación del objeto HTTPClient

Una aplicación de .NET MAUI puede consumir un servicio web basado en REST mediante el envío de solicitudes al servicio web con la clase **HttpClient**. Esta clase proporciona funcionalidad para enviar solicitudes HTTP y recibir respuestas HTTP de un recurso identificado por una URI. Cada solicitud se envía como una operación asincrónica.

```
public class RestService : IRestService
{
    HttpClient _client;
    JsonSerializerOptions _serializerOptions;

    public List<TodoItem> Items { get; private set; }

    public RestService()
    {
        _client = new HttpClient();
        _serializerOptions = new JsonSerializerOptions
        {
            PropertyNamingPolicy = JsonNamingPolicy.CamelCase,
            WriteIndented = true
        };
    }
    ...
}
```

Recuperación de datos

El método HttpClient.**GetAsync** se usa para enviar una solicitud GET al servicio web especificado por una URI y, a continuación, recibir la respuesta del servicio web:

```
public async Task<List<TodoItem>> RefreshDataAsync()
{
    Items = new List<TodoItem>();

    Uri uri = new Uri(string.Format(Constants.RestUrl, string.Empty));
    try
    {
        HttpResponseMessage response = await _client.GetAsync(uri);
        if (response.IsSuccessStatusCode)
        {
            string content = await response.Content.ReadAsStringAsync();
            Items = JsonSerializer.Deserialize<List<TodoItem>>(content, _serializerOptions);
        }
    }
    catch (Exception ex)
    {
        Debug.WriteLine(@"\tERROR {0}", ex.Message);
    }

    return Items;
}
```

Recuperación de datos

Los datos se reciben del servicio web como un objeto **HttpResponseMessage**. Contiene información sobre la respuesta, incluido el código de estado, los encabezados y cualquier contenido del cuerpo. El servicio REST envía un código de estado HTTP en su respuesta, que se puede obtener de la propiedad HttpResponseMessage.**IsSuccessStatusCode**, para indicar si la solicitud HTTP se realizó correctamente o no.

Si la operación HTTP se realizó correctamente, se lee el contenido de la respuesta. La propiedad HttpResponseMessage.**Content** representa el contenido de la respuesta y es de tipo **HttpContent**. La clase HttpContent representa el cuerpo HTTP y los encabezados de contenido, como Content-Type y Content-Encoding. A continuación, el contenido se lee en un string mediante el método HttpContent.**ReadAsStringAsync**. A continuación, se deserializa de JSON a objeto o listado de objetos.

Data Serialization

- Los objetos .NET deben convertirse en bytes para poder enviarlos o recibirlos mediante la red, este proceso se denomina serialización.
- La serialización ocurre cada vez que nos comunicamos a través de una red, independientemente de la tecnología que se utilice para transferir información.

Crear datos

El método HttpClient.**PostAsync** se usa para enviar una solicitud POST al servicio web especificado por una URI y, a continuación, recibir la respuesta del servicio web:

```
public async Task SaveTodoItemAsync(TodoItem item, bool isNewItem = false)
{
    Uri uri = new Uri(string.Format(Constants.RestUrl, string.Empty));

    try
    {
        string json = JsonSerializer.Serialize<TodoItem>(item, _serializerOptions);
        StringContent content = new StringContent(json, Encoding.UTF8, "application/json");

        HttpResponseMessage response = null;
        if (isNewItem)
            response = await _client.PostAsync(uri, content);
        else
            response = await _client.PutAsync(uri, content);

        if (response.IsSuccessStatusCode)
            Debug.WriteLine(@"\tTodoItem successfully saved.");
    }
    catch (Exception ex)
    {
        Debug.WriteLine(@"\tERROR {0}", ex.Message);
    }
}
```

Crear datos

En el ejemplo anterior, la instancia TodoItem se serializa en JSON para enviar al servicio web. A continuación, esta carga se inserta en el cuerpo del contenido HTTP que se enviará al servicio web antes de que se realice la solicitud con el método PostAsync.

El servicio REST envía un código de estado HTTP en su respuesta, que se puede obtener de la propiedad HttpResponseMessage.**IsSuccessStatusCode**, para indicar si la solicitud HTTP se realizó correctamente o no. Las respuestas típicas para esta operación son:

- **201** (CREATED): la solicitud dio lugar a que se creara un nuevo recurso antes de enviar la respuesta.
- **400** (SOLICITUD INCORRECTA): el servidor no entiende la solicitud.
- **409** (CONFLICT): no se pudo realizar la solicitud debido a un conflicto en el servidor.

Actualización de datos

El método HttpClient.**PutAsync** se usa para enviar una solicitud PUT al servicio web especificado por el URI y, a continuación, recibir la respuesta del servicio web:

```
public async Task SaveTodoItemAsync(TodoItem item, bool isNewItem = false)
{
  ...
  response = await _client.PutAsync(uri, content);
  ...
}
```

La operación del método **PutAsync** es idéntica al método PostAsync que se usa para crear datos en el servicio web. Sin embargo, las posibles respuestas enviadas desde el servicio web difieren.

El servicio REST envía un código del estado HTTP en su respuesta, que se puede obtener de la propiedad HttpResponseMessage.**IsSuccessStatusCode**, para indicar si la solicitud HTTP se realizó correctamente o no. Las respuestas típicas para esta operación son:

- **204** (SIN CONTENIDO): la solicitud se ha procesado correctamente y la respuesta está en blanco intencionadamente.
- **400** (SOLICITUD INCORRECTA): el servidor no entiende la solicitud.
- **404** (NO ENCONTRADO): el recurso solicitado no existe en el servidor.

Eliminación de datos

El método HttpClient.**DeleteAsync** se usa para enviar una solicitud DELETE al servicio web especificado por una URI y, a continuación, recibir la respuesta del servicio web:

```
public async Task DeleteTodoItemAsync(string id)
{
    Uri uri = new Uri(string.Format(Constants.RestUrl, id));

    try
    {
        HttpResponseMessage response = await _client.DeleteAsync(uri);
        if (response.IsSuccessStatusCode)
            Debug.WriteLine(@"\tTodoItem successfully deleted.");
    }
    catch (Exception ex)
    {
        Debug.WriteLine(@"\tERROR {0}", ex.Message);
    }
}
```

El servicio REST envía un código de estado HTTP en su respuesta, que se puede obtener de la propiedad HttpResponseMessage.**IsSuccessStatusCode**, para indicar si la solicitud HTTP se realizó correctamente o no. Las respuestas típicas para esta operación son:

- **204** (SIN CONTENIDO): la solicitud se ha procesado correctamente y la respuesta está en blanco intencionadamente.
- **400** (SOLICITUD INCORRECTA): el servidor no entiende la solicitud.
- **404** (NO ENCONTRADO): el recurso solicitado no existe en el servidor.

Desarrollo local

Al desarrollar un servicio web REST localmente con ASP.NET Core API web, se puede depurar el servicio web y la aplicación .NET MAUI al mismo tiempo. En este escenario, para consumir el servicio web a través de HTTP desde emuladores de Android y simuladores de iOS, se debe habilitar el tráfico **HTTP de texto no cifrado**.