Artículo
Ricardo Paiva · 29 nov, 2021 Lectura de 14 min

Open Exchange

Cómo transferir archivos a través de REST para almacenar en una propiedad. Parte 2

En la primera parte de esta serie de artículos, hablamos sobre cómo leer un fragmento "grande" de datos del contenido sin procesar de un método HTTP POST y guardarlo en una base de datos como una propiedad de flujo de una clase. Ahora veremos cómo guardar esos datos y metadatos en formato JSON.

Desafortunadamente, Advanced REST Client no permite configurar objetos JSON con datos binarios como valor de una clave (o quizá simplemente no he descubierto cómo hacerlo), así que decidí escribir un cliente simple en ObjectScript para enviar datos al servidor.

Creé una nueva clase llamada RestTransfer.Client y le añadí los parámetros Server = "localhost" y Port = 52773 para describir mi servidor web. Y creé un método de clases GetLink en el que creo una nueva instancia de la clase %Net.HttpRequest y establezco sus propiedades con los parámetros mencionados anteriormente.

Para enviar la solicitud POST al servidor, creé un método de clase SendFileDirect, que lee el archivo que quiero enviar al servidor y escribe su contenido en la propiedad EntityBody de mi solicitud.

Después de esto, llamo al método Post("/RestTransfer/file") y, si se completa correctamente, la respuesta estará en la propiedad HttpResponse.

Para ver el resultado devuelto por el servidor, llamo al método OutputToDevice de la respuesta.

Este es el método de la clase:

```
Class RestTransfer.Client
{
  Parameter Server = "localhost";
  Parameter Port = 52773;
  Parameter Https = 0;

  ClassMethod GetLink() As %Net.HttpRequest
  {
      set request = ##class(%Net.HttpRequest).%New()
      set request.Server = ..#Server
      set request.Port = ..#Port
      set request.ContentType = "application/octet-stream"
      quit request
  }

  ClassMethod SendFileDirect(aFileName) As %Status
  {
    set sc = $$$OK
    set request = ..GetLink()
    set s = ##class(%Stream.FileBinary).%New()
```

```
    set s.Filename = aFileName
    While 's.AtEnd
    {
      do request.EntityBody.Write(s.Read(.len, .sc))
      Quit:$System.Status.IsError(sc)
    }
    Quit:$System.Status.IsError(sc)

    set sc = request.Post("/RestTransfer/file")
    Quit:$System.Status.IsError(sc)
    set response=request.HttpResponse
    do response.OutputToDevice()
    Quit sc
  }
}
```

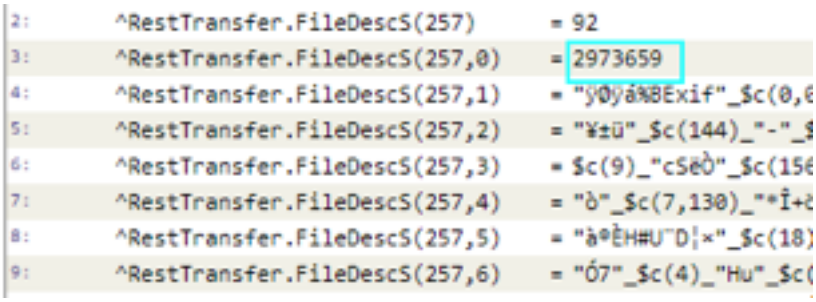
Podemos llamar a este método para transferir los mismos archivos que se utilizaron en el artículo anterior:

```
 do ##class(RestTransfer.Client).SendFileDirect("D:\Downloads \2020_1012_114732_020.J
PG")
 do ##class(RestTransfer.Client).SendFileDirect("D:\Downloads\Archive.xml")
 do ##class(RestTransfer.Client).SendFileDirect("D:\Downloads\arc-setup.exe")
```

Observaremos las respuestas del servidor para cada archivo:

```
HTTP/1.1 200 OK
CACHE-CONTROL: no-cache
CONTENT-LENGTH: 15
CONTENT-TYPE: text/html; charset=utf-8
DATE: Thu, 05 Nov 2020 15:13:23 GMT
EXPIRES: Thu, 29 Oct 1998 17:04:19 GMT
PRAGMA: no-cache
SERVER: Apache
{"Status":"OK"}
```

Y tendremos los mismos datos en los *globals* que antes:



```
96:   ^RestTransfer.FileDescS(258)     = 1
97:   ^RestTransfer.FileDescS(258,0)   = 8011
98:   ^RestTransfer.FileDescS(258,1)   = "<?xml version=""1.0"" encoding=""UTF-8""?>"_$c(13,10)_"<Export
99:   ^RestTransfer.FileDescS(259)     = 3311
100:  ^RestTransfer.FileDescS(259,0)   = 108112056
101:  ^RestTransfer.FileDescS(259,1)   = "MZ"_$c(144,0,3,0,0,0,4,0,0,0)_"ÿÿ"_$c(0,0)_"."_$c(0,0,0,0,0,0,0
102:  ^RestTransfer.FileDescS(259,2)   = "leges"_$c(0)_"P"_$c(1)_"LookupPrivilegeValueW"_$c(0)_"¬"_$c(1)_"
103:  ^RestTransfer.FileDescS(259,3)   = "Û¡"_$c(22)_"ÿÛ¡"_$c(22)_"ÿÛ¡"_$c(22)_"ÿÛ¡"_$c(22)_"ÿÛ¡"_$c(22)_"
104:  ^RestTransfer.FileDescS(259,4)   = "ÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿÿ
105:  ^RestTransfer.FileDescS(259,5)   = "Û¡"_$c(22)_"ÿÛ¡"_$c(22)_"ÿÛ¡"_$c(22)_"ÿÛ¡"_$c(22)_"ÿÛ¡"_$c(22)_"
```

Esto significa que nuestro cliente funciona y ahora lo podemos expandir para enviar JSON al servidor.

Introducción a JSON

JSON es un formato ligero para almacenar y transportar datos en el que los datos están en pares nombre/valor separados por comas. En este caso, el JSON será algo parecido a esto:

```
{
  "Name": "test.txt",
  "File": "Hello, world!"
}
```

IRIS tiene varias clases que permiten trabajar con JSON. Yo utilizaré la siguientes:

- %JSON.Adaptor ofrece una forma de serializar un objeto habilitado para JSON como un documento JSON y viceversa.
- %Library.DynamicObject ofrece una forma de ensamblar de forma dinámica datos que se pueden transferir cómodamente entre un cliente web y un servidor.

Puedes encontrar más información sobre estas y otras clases que te permiten trabajar con JSON en la sección *Class Reference* de la documentación y en las guías para desarrollar aplicaciones.

La principal ventaja de estas clases es que ofrecen una forma sencilla de crear JSON a partir de un objeto IRIS o de crear un objeto a partir de JSON.

Voy a mostrar dos enfoques sobre cómo se puede trabajar con JSON para enviar/recibir archivos, en concreto uno que utiliza %Library.DynamicObject para crear un objeto en el lado del cliente y serializarlo en un documento JSON y %JSON.Adaptor para deserializarlo de nuevo a RestTransfer.FileDesc en el lado del servidor, y otro en el que desarrollo de forma manual una cadena para enviar y analizar en el lado del servidor. Para que sea más legible, crearé dos métodos diferentes en el servidor para cada enfoque.

Por ahora, nos centraremos en la primera.

Cómo crear JSON con IRIS

Para empezar, debemos heredar la clase RestTransfer.FileDesc de %JSON.Adaptor. Esto nos permite utilizar el método de instancia %JSONImport() para deserializar un documento JSON directamente en

un objeto. Ahora, esta clase tendrá el siguiente aspecto:

```
Class RestTransfer.FileDesc Extends (%Persistent, %JSON.Adaptor)
{
  Property File As %Stream.GlobalBinary;
  Property Name As %String;
}
```

Añadamos una nueva ruta a UrlMap en el broker de la clase:

```
<Route Url="/jsons" Method="POST" Call="InsertJSONSmall"/>
```

Esto especifica que, cuando el servicio recibe un comando POST con la URL /RestTransfer/jsons, debería llamar al método de clase InsertJSONSmall.

Este método espera recibir un texto en formato JSON con dos pares clave-valor donde los contenidos del fichero serán inferiores a la longitud máxima de una cadena. Definiré las propiedades Name y File del objeto, lo guardaré en la base de datos y devolveré el estado y un mensaje con formato JSON que indicará si es correcto o es un error.

Este es el método de la clase.

```
ClassMethod InsertJSONSmall() As %Status
{
  Set result={}
  Set st=0

  set f = ##class(RestTransfer.FileDesc).%New()
  if (f = $$$NULLOREF) {
   do result.%Set("Message", "Couldn't create an instance of class")
  } else {
     set st = f.%JSONImport(%request.Content)
     If $$$ISOK(st) {
        set st = f.%Save()
        If $$$ISOK(st) {
           do result.%Set("Status","OK")
        } else {
           do result.%Set("Message",$system.Status.GetOneErrorText(st))
        }
     } else {
        do result.%Set("Message",$system.Status.GetOneErrorText(st))
     }
  }
  write result.%ToJSON()
  Quit st
}
```

¿Qué hace este método? Recibe la propiedad Content del objeto %request y, usando el método %JSONImport heredado de la clase %JSON.Adaptor, la convierte en un objeto RestTransfer.FileDesc.

Si se producen problemas durante la conversión, configuramos un JSON con la descripción del error:

```
{"Message",$system.Status.GetOneErrorText(st)}
```

De lo contrario, guardamos el objeto. Si se guarda sin problemas, creamos un mensaje JSON {"Status","OK"}. En caso contrario, un mensaje JSON con la descripción del error.

Finalmente, escribimos el código JSON en una respuesta y devolvemos el estado st.

En el lado del cliente añadí un nuevo método de clase SendFile para enviar archivos al servidor. El código es muy similar al de SendFileDirect pero, en vez de escribir los contenidos del archivo directamente en la propiedad EntityBody, creo una nueva instancia de la clase %Library.DynamicObject, establezco su propiedad Name igual al nombre del archivo, y codifico y copio los contenidos del archivo en la propiedad File.

Para codificar los contenidos del archivo utilizo el método Base64EncodeStream() propuesto por Vitaliy Serdtsev.

Después, usando el método %ToJSON de la clase %Library.DynamicObject, serializo mi objeto en un documento JSON, lo escribo en el contenido de la solicitud y llamo al método Post("/RestTransfer/jsons").

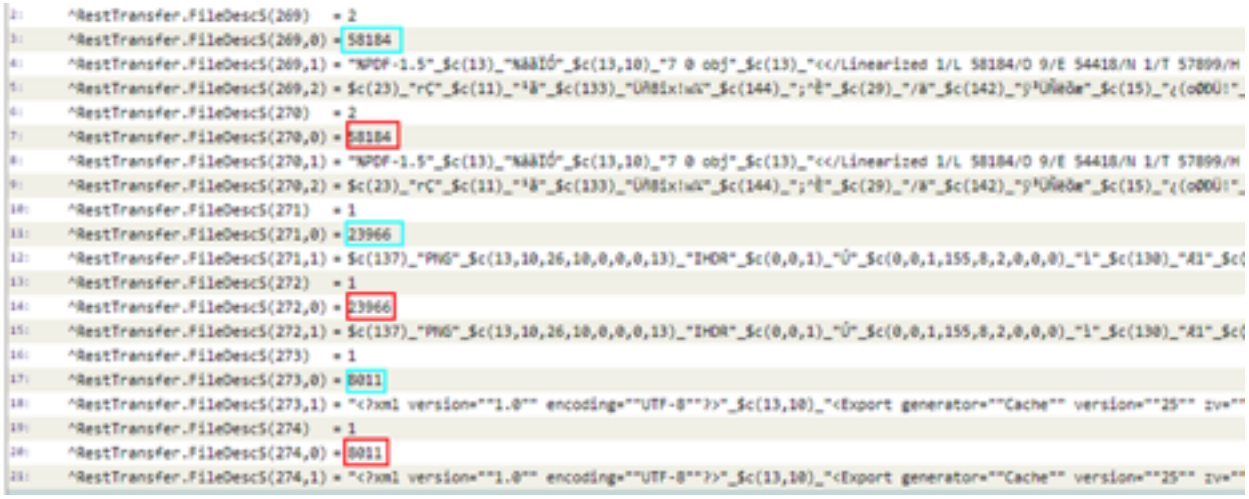
Este es el método de la clase:

```
ClassMethod SendFile(aFileName) As %Status
{
  Set sc = $$$OK
  Set request = ..GetLink()
  set s = ##class(%Stream.FileBinary).%New()
  set s.Filename = aFileName
  set p = {}
  set p.Name = s.Filename
  set sc = ..Base64EncodeStream(s, .t)
  Quit:$System.Status.IsError(sc)

  While 't.AtEnd {
    set p.File = p.File_t.Read(.len, .sc)
    Quit:$System.Status.IsError(sc)
  }
  do p.%ToJSON(request.EntityBody)
  Quit:$System.Status.IsError(sc)

  set sc = request.Post("/RestTransfer/jsons")
  Quit:$System.Status.IsError(sc)

  Set response=request.HttpResponse
  do response.OutputToDevice()
  Quit sc
}
```

Llamamos a este método y al método SendFileDirect para transferir varios archivos pequeños:

```
 do ##class(RestTransfer.Client).SendFile("D:\Downloads\Outline Template.pdf")
 do ##class(RestTransfer.Client).SendFileDirect("D:\Downloads\Outline Template.pdf")

 do ##class(RestTransfer.Client).SendFile("D:\Downloads\pic3.png")
 do ##class(RestTransfer.Client).SendFileDirect("D:\Downloads\pic3.png")
 do ##class(RestTransfer.Client).SendFile("D:\Downloads\Archive (1).xml")
 do ##class(RestTransfer.Client).SendFileDirect("D:\Downloads\Archive (1).xml")
```

Obtendremos los siguientes resultados:



Como puedes observar, las longitudes son las mismas y, si guardamos esos archivos en un disco duro, veremos que no han cambiado.

Configuración manual de JSON

Ahora nos centraremos en el segundo enfoque, que consiste en configurar el JSON manualmente. Para hacerlo, añadiremos una nueva ruta a UrlMap en el broker de la clase:

```
<Route Url="/json" Method="POST" Call="InsertJSON"/>
```

Esto especifica que cuando el servicio recibe un comando POST con la URL /RestTransfer/json, debería llamar al método de clase InsertJSON. En este método, espero recibir el mismo JSON, pero no impongo ningún límite a la longitud del archivo. Este es el método de la clase:

```
ClassMethod InsertJSON() As %Status
{
  Set result={}
  Set st=0

  set t = ##class(%Stream.TmpBinary).%New()
  While '%request.Content.AtEnd
  {
    set len = 32000
    set temp = %request.Content.Read(.len, .sc)
    set:len<32000 temp = $extract(temp,1,*-2)
    set st = t.Write($ZCONVERT(temp, "I", "RAW"))

  }
  do t.Rewind()

  set f = ##class(RestTransfer.FileDesc).%New()
  if (f = $$$NULLOREF)
  {
    do result.%Set("Message", "Couldn't create an instance of class")
  } else {
```

```
    set str = t.Read()
    set pos = $LOCATE(str,""",")
    set f.Name = $extract(str, 10, pos-1)
    do f.File.Write($extract(str, pos+11, *))
    While 't.AtEnd {
      do f.File.Write(t.Read(.len, .sc))
    }

    If $$$ISOK(st)
    {
      set st = f.%Save()
      If $$$ISOK(st)
      {
        do result.%Set("Status","OK")
      } else {
        do result.%Set("Message",$system.Status.GetOneErrorText(st))
      }
    } else {
       do result.%Set("Message",$system.Status.GetOneErrorText(st))
    }
  }
  write result.%ToJSON()
  Quit st
}
```

¿Qué hace este método? Primero, creo una nueva instancia de un flujo temporal, %Stream.TmpBinary, y copio en él el contenido de la solicitud.

Como voy a trabajar con ella como una cadena, necesito deshacerme de las comillas finales (") y de las llaves (}). Para hacerlo, en el último fragmento del flujo dejo fuera los dos últimos caracteres: $extract(temp,1,*-2).

A la vez, convierto mi cadena usando la tabla de traducción "RAW": $ZCONVERT(temp, "I", "RAW").

Después creo una nueva instancia de mi clase RestTransfer.FileDesc y hago las mismas verificaciones que en los otros métodos. Ya conozco la estructura de mi cadena, así que extraigo el nombre del archivo y el archivo en sí y las defino en las propiedades correspondientes.

En el lado del cliente, modifiqué mi método de clase SendFile y, antes de crear el JSON, verifico la longitud del archivo. Si es inferior a los 2 000 000 de bytes (límite aparente del método %JSONImport), llamo a Post("/RestTransfer/jsons"). De lo contrario, llamo a Post("/RestTransfer/json").

Ahora el método se ve así:

```
ClassMethod SendFile(aFileName) As %Status
{
  Set sc = $$$OK
  Set request = ..GetLink()
  set s = ##class(%Stream.FileBinary).%New()
  set s.Filename = aFileName
  if s.Size > 2000000 //3641144 max length of the string in IRIS
  {
    do request.EntityBody.Write("{""Name"":""_s.Filename_""", ""File"":""")
    While 's.AtEnd
    {
      set temp = s.Read(.len, .sc)
```

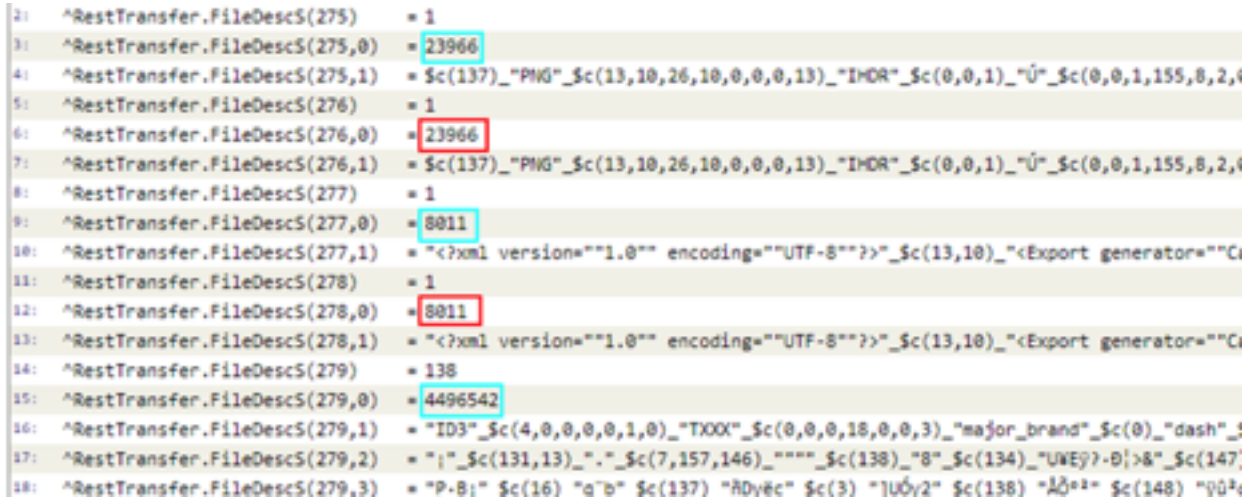
```
      do request.EntityBody.Write($ZCONVERT(temp, "O", "RAW"))
      Quit:$System.Status.IsError(sc)
    }

    do request.EntityBody.Write("""}")
    set sc = request.Post("/RestTransfer/json")
  } else {
    set p = {}
    set p.Name = s.Filename
    set sc = ..Base64EncodeStream(s, .t)
    Quit:$System.Status.IsError(sc)
    While 't.AtEnd {
        set p.File = p.File_t.Read(.len, .sc)
        Quit:$System.Status.IsError(sc)
    }
    do p.%ToJSON(request.EntityBody)
    Quit:$System.Status.IsError(sc)
    set sc = request.Post("/RestTransfer/jsons")

  }

  Quit:$System.Status.IsError(sc)
  Set response=request.HttpResponse
  do response.OutputToDevice()
  Quit sc
}
```

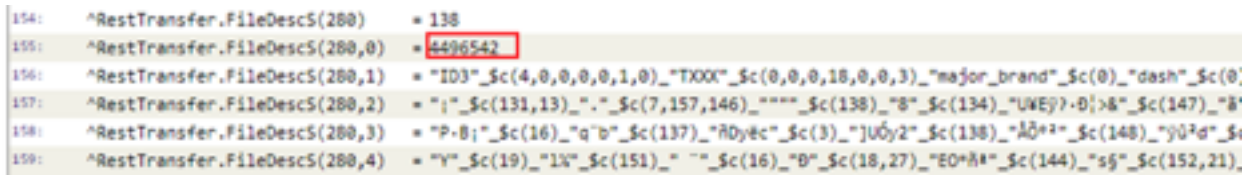
Para llamar al segundo método, creo la cadena con JSON de forma manual. Y para transferir el archivo en sí, en el lado del cliente también convierto los contenidos mediante la tabla de traducción "RAW": ($ZCONVERT(temp, "O", "RAW").

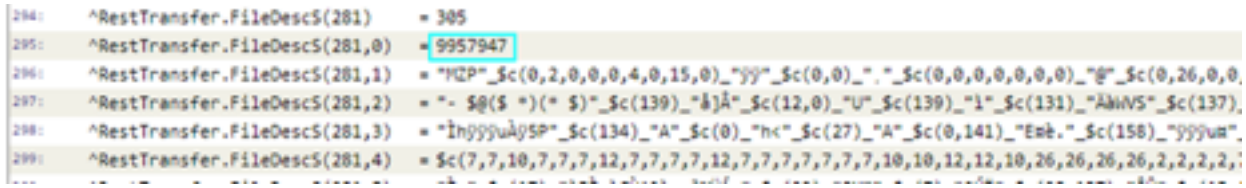
Ahora llamamos a este método y al método SendFileDirect para transferir varios archivos de diferentes tamaños:

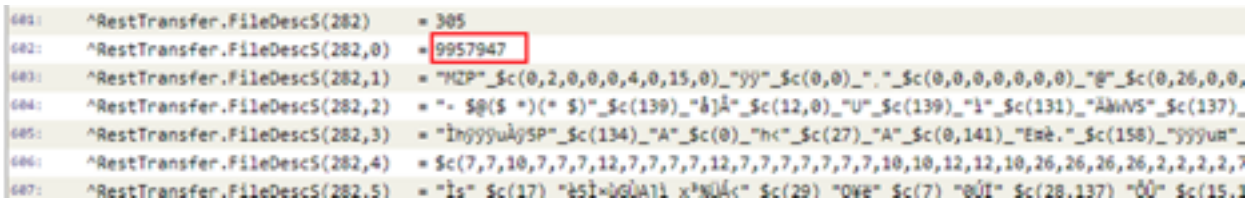
```
 do ##class(RestTransfer.Client).SendFile(“D:\Downloads\pic3.png”)
 do ##class(RestTransfer.Client).SendFileDirect(“D:\Downloads\pic3.png”)
 do ##class(RestTransfer.Client).SendFile(“D:\Downloads\Archive (1).xml”)
 do ##class(RestTransfer.Client).SendFileDirect(“D:\Downloads\Archive (1).xml”)
 do ##class(RestTransfer.Client).SendFile(“D:\Downloads\Imagine Dragons-
Thunder.mp3”)
 do ##class(RestTransfer.Client).SendFileDirect(“D:\Downloads\Imagine Dragons-
Thunder.mp3”)
 do ##class(RestTransfer.Client).SendFile(“D:\Downloads\ffmpeg-win-2.2.2.exe”)
 do ##class(RestTransfer.Client).SendFileDirect(“D:\Downloads\ffmpeg-win-2.2.2.exe”)
```

Obtendremos los siguientes resultados:









Podremos ver que las longitudes son las mismas por lo que todo funciona como se supone.

Conclusión

De nuevo, puedes leer más sobre la creación de servicios REST en la documentación. El código de ejemplo para ambos enfoques se encuentra en GitHub y en InterSystems Open Exchange.

Sobre la transferencia de resultados del módulo TWAIN que se comenta en el primer artículo de esta serie, depende del tamaño de los datos que recibas. Si la máxima resolución será limitada y los archivos serán pequeños, puedes utilizar las clases del sistema %JSON.Adaptor y %Library.DynamicObject. Pero para estar seguros, es mejor enviar un archivo directamente al cuerpo de un comando POST o configurar el JSON manualmente. También puede ser una buena idea utilizar solicitudes que abarquen varios archivos y dividirlos en varias partes, enviarlos por separado y volver a consolidarlos en un solo archivo en el lado del servidor.

En cualquier caso, si tienes alguna pregunta o sugerencia, puedes dejarla en la sección de comentarios.

#API REST #InterSystems IRIS
Ir a la aplicación en InterSystems Open Exchange