API RESTful

Artículo
Kurro Lopez · Jun 18, 2019 Lectura de 14 min

API RESTful

¡Hola Comunidad!

Les dejo una guía para principiantes sobre todo lo que necesitan saber del diseño y la documentación de la Interfaz de programación para aplicaciones (API) RESTful. Mediante este ejemplo aprenderán algunos de los patrones más comunes de la API RESTful.

¡Espero que les sea útil!

Antes de leer el artículo

Necesita saber lo siguiente:

- Cómo crear un servicio web RESTful en Ensemble
- Cómo consumir el servicio web RESTful en Ensemble
- Cómo aprobar los parámetros del servicio
- Cómo devolver los resultados del servicio

¿Qué es el servicio de una API?

¿Qué es una Interfaz de programación para aplicaciones (API)? ¿Es algo que puede materializarse? ¿Es una unidad de programación única? ¿Para qué sirve una API? Desde mi punto de vista, una API es algo que se determina indirectamente por el código del programa. Pero la definición completa de la API se proporciona por un contenedor (controlado por la configuración de la implementación) que funciona con el ejecutable de un programa. De modo que, prefiero definir a la API como la descripción pública de un servicio. La descripción puede ser comprensible ya sea para las personas o solo para los robots, o para ambos. La función primordial de una API es compartir la información sobre un servicio, con aquellos que están a punto de consumirlo. Una API brinda información sobre cuál es el servicio, el contexto en el que puede utilizarse, cuáles son sus funciones, qué estructuras de datos administra, etc.

En los viejos tiempos, la “documentación de un programa” era más o menos "un mal necesario". Los lenguajes de programación modernos fueron obligados a utilizar una especie de documentación al introducir declaraciones en el programa fuente. Aunque las declaraciones se hicieron en documentos que solo eran legibles para los “robots”, con la ayuda de ciertas herramientas (runoff, Java doc....) se podía extraer la información y darle formato para que fuera comprensible para las personas. Incluso si no se agregaba una sola línea del verdadero documento a la fuente, estas herramientas eran capaces de producir alguna cantidad mínima de texto.

¿Hubo algún cambio en los tiempos actuales? En realidad no. El servicio que brindan las API sigue siendo

una abstracción, es decir, lo que la gente entiende por la recopilación de la información necesaria para utilizar correctamente una pieza funcional de software informático. Existen lenguajes para formalizar la definición de las API, como Web Services Description Language (WSDL). Desafortunadamente, este tipo de lenguajes tienen un uso limitado. Por ejemplo, no es que WSDL no sea lo suficientemente capaz para expresar lo que quiere decir una API RESTful, el problema es que existe una falta de correspondencia que no es técnica. (¿Qué objetivo tiene expresar una estructura JSON en XML?) Al final, en realidad no existe un lenguaje estándar para REST como el WSDL para los servicios web que ofrece SOAP. Lo cual es una lástima. ¿No es así?

No importa. De todos modos, primero necesitamos entender lo que registra una API.

¿De qué está hecha una API?

Los atributos principales de un servicio son:

- La ubicación del servicio. Es la ruta para la URL raíz del servicio, como http://localhost:57774/csp/msa/person.
- Los métodos del servicio. Son las funciones de un servicio. Un método se define por la combinación del verbo en el encabezado HTTP (GET, POST, PUT...) y los parámetros adicionales del tipo de ruta.
- Los parámetros que acepta el método. Son una lista de los parámetros de acuerdo a su tipo. El tipo puede ser path, para los parámetros que se incluyen en la ruta de la URL, query para las consultas hacia la URL codificada, form para los datos del formulario, o content para el cuerpo del mensaje HTTP.
- Estado de la respuesta. Es el campo correspondiente al estado del encabezado de respuesta HTTP. Por cada método de servicio, podría haber varios códigos para el estado de la respuesta. El número depende del método del servicio y del nivel de detalle que tenga la administración de las excepciones.
- Contenido de las respuestas. Es el contenido esperado por cada código de estado. El formato puede ser diferente de acuerdo al estado del código. Por ejemplo, cuando se completa con éxito una solicitud, se espera un objeto JSON serializado. En caso de que exista un error en el servidor (500) se enviará una explicación en texto sin formato.

Ejemplo de una API

En primer lugar, hagamos un intento por describir lo que tratamos de lograr. Vamos a construir un servicio muy sencillo. Un registro para el servicio. Esto tendrá la función de administrar los recursos que sean del mismo tipo. Por ejemplo, las personas.

La estructura es muy sencilla: un nombre, una fecha y un lugar de nacimiento, el apellido de soltera de la madre y la generación un ID de registro único interno. El lugar tiene una estructura como: país, ciudad.

Nos gustaría insertar un nuevo historial en nuestro registro (actualizar la entrada del registro completa), actualizar los atributos individuales de una entrada (actualizar un atributo), eliminar una entrada, obtener una sola entrada por medio del ID de registro, consultar una lista de los ID de registro con base en la coincidencia de los atributos.

También queremos alguna función para el servicio: inicializar un registro, y añadir algunos historiales para probarlo.

Desde el mundo exterior nos gustaría tener la dirección http://localhost:57774/csp/msa/person como una ubicación para el servicio.

Añada una nueva entrada mediante PUT para la ubicación del servicio. Envíe el historial de registro como un contenido. En la respuesta se espera la entrada completa con un ID de registro.

Realice una actualización mediante POST. La URL se completa con el ID de registro de la entrada que debe actualizarse. Los atributos que deben actualizarse se envían como los datos del formulario.

El método GET se utiliza para recuperar datos. Si la ruta de la URL termina con un ID, entonces las entradas en el registro se identifican por el ID de la respuesta. Si no se encuentra ningún ID, pero existe una consulta en la URL, devolverá una lista de los ID. Por ejemplo, http://localhost:57774/csp/msa/person/12A33 devuelve la entrada 12A33. Los pares de valores query key son las cláusulas que corresponden con los atributos que se utilizaron de forma interna para seleccionar las entradas. Por ejemplo, http://localhost:57774/csp/msa/person?name=Hahn%20Istvan&dob=1961 devuelve una lista de las personas que nacieron en 1961 con el nombre de Hahn Istvan.

DELETE hace eliminaciones.

POST en http://localhost:57774/csp/msa/person/_init inicializará el registro.

POST en http://localhost:57774/csp/msa/person/_populate/100 carga 100 entradas de prueba.

Documentación de la API

En la siguiente sección se muestra un ejemplo de cómo puede documentarse el servicio de una API. Recuerde que ni la estructura ni el contenido se estandarizaron. Este es solo un ejemplo.

Traté de elaborar la documentación para una "herramienta agnóstica". En el mercado existen diversas herramientas para realizar la documentación. Algunas de ellas hacen su trabajo casi tan bien como deberían. La intención de esta sección que tenga una idea sobre la complejidad de la documentación de una API, si la realiza con un editor de texto.

Recurso: person

n servicio genérico para administrar los recursos de tipo person. Person tiene un conjunto mínimo de atributos. Básicamente son los datos demográficos y una ID de registro.

Ubicación: http://localhost:57774/csp/msa/person

Método:

Obtenga un solo recurso con base en el ID del recurso.

Verbo: GET

Parámetros:

Nombre	Tipo	Tipo de dato	Comentarios
1	Path	Resource ID	Es el único ID del recurso que podrá recuperarse.

Respuesta:

Estado	Tipo de respuesta	Comentarios

200	Person	Se encontró el historial.
204	None	No existe ningún historial con el ID del recurso.
401	None	Acceso no autorizado. El recurso necesita la acreditación del usuario en el encabezado.
403	None	Prohibido. El usuario no está autorizado para acceder al recurso.
500	Error	Error interno del servidor.
501	Error	No se implementó el método solicitado.
503	Error	El servicio no está disponible temporalmente.

Método:

Obtenga una lista de los ID que coincidan basados los query, que no son únicos. Este método utiliza la parte de la query para construir la query de tipo string. Los pares de valores query key/ se convierten en los pares de valores de la columna name/.

Verbo: GET

Parámetros:

Nombre	Tipo	Tipo de dato	Comentarios
name	query	string	Criterio de búsqueda.
motherMaidenName	Query	String	
dob	Query	Date	
birthPlaceCounty	Query	String	
birthPlaceCity	Query	string	

Respuesta:

Estado	Tipo de respuesta	Comentarios
200	Person	Se encontró el historial.
204	None	No se encontraron coincidencias en el historial.

401	None	Acceso no autorizado. El recurso necesita la acreditación del usuario en el encabezado.
403	None	Prohibido. El usuario no está autorizado para acceder al recurso.
500	Error	Error interno del servidor.
501	Error	No se implementó el método solicitado.
503	Error	El servicio no está disponible temporalmente.

Método:

Elimina una entrada del registro.

Verbo: DELETE

Parámetros:

Nombre	Tipo	Tipo de dato	Comentarios
1	path	string	ID de registro único.

Respuesta:

Estado	Tipo de respuesta	Comentarios
200	Person	Se eliminó el historial.
401	None	Acceso no autorizado. El recurso necesita la acreditación del usuario en el encabezado.
403	None	Prohibido. El usuario no está autorizado para acceder al recurso.
500	Error	Error interno del servidor.
501	Error	No se implementó el método solicitado.
503	Error	El servicio no está disponible temporalmente.

Método:

Agregar o actualizar una entrada en el registro.

Verbo: PUT

Parámetros:

Nombre	Tipo	Tipo de dato	Comentarios
None	content	JSON	Un objeto serializado en formato JSON.

Respuesta:

Estado	Tipo de respuesta	Comentarios
200	Person	Es la entrada con el ID del recurso que se generó, ya sea que se haya añadido de forma reciente o se haya actualizado mediante el registro del servicio.
401	None	Acceso no autorizado. El recurso necesita la acreditación del usuario en el encabezado.
403	None	Prohibido. El usuario no está autorizado para acceder al recurso.
500	Error	Error interno del servidor.
501	Error	No se implementó el método solicitado.
503	Error	El servicio no está disponible temporalmente.

Método:

Actualizar los atributos individuales de una entrada del registro.

Verbo: POST

Parámetros:

Nombre	Tipo	Tipo de dato	Comentarios	
1	Path	String	Es el ID del recurso	

			de la entrada que debe actualizarse.
name	Form	string	El nuevo valor del atributo
motherMaidenName	Form	String	
dob	Form	Date	
birthPlaceCounty	Form	String	
birthPlaceCity	Form	string	

Respuesta:

Estado	Tipo de respuesta	Comentarios
200	Person	Se actualizó el historial.
204	None	No existe ningún historial con el ID del recurso.
401	None	Acceso no autorizado. El recurso necesita la acreditación del usuario en el encabezado.
403	None	Prohibido. El usuario no está autorizado para acceder al recurso.
500	Error	Error interno del servidor.
501	Error	No se implementó el método solicitado.
503	Error	El servicio no está disponible temporalmente.

Método:

Inicializar el registro.

Verbo: POST

Parámetros:

Nombre	Tipo	Tipo de dato	Comentarios
_init	Path		

Respuesta:

Estado	Tipo de respuesta	Comentarios
200	None	Se inicializó.
401	None	Acceso no autorizado. El recurso necesita la acreditación del usuario en el encabezado.
403	None	Prohibido. El usuario no está autorizado para acceder al recurso.
500	Error	Error interno del servidor.
501	Error	No se implementó el método solicitado.
503	Error	El servicio no está disponible temporalmente.

Método:

Añadir los datos de la prueba.

Verbo: POST

Parámetros:

Nombre	Tipo	Tipo de dato	Comentarios
_populate	Path		
2	Path	Numeric	Número de entradas que deben añadirse.

Respuesta:

Estado	Tipo de respuesta	Comentarios
200	None	Se inicializó.
401	None	Acceso no autorizado. El recurso necesita la acreditación del usuario en el encabezado.
403	None	Prohibido. El usuario no está autorizado para acceder al recurso.

500	Error	Error interno del servidor.
501	Error	No se implementó el método solicitado.
503	Error	El servicio no está disponible temporalmente.

Las estructuras de los datos:

Person

Nombre	Tipo	Marca	Comentarios
ID	RegistryID	R	Se generó el ID de registro.
Name	String	R	Nombre de la persona en su lengua materna como aparece en el formulario.
DOB	Date	R	Fecha de nacimiento.
BirthPlace	BirthPlace	O	Lugar de nacimiento.
MotherMaidenName	String	O	Apellido de soltera de la madre.

BirthPlace

Nombre	Tipo	Marca	Comentarios
Country	String	O	Código del país.
City	String	R	Nombre de la ciudad

Error

Nombre	Tipo	Marca	Comentarios
Code	String	R	Error en el código
Text	String	O	Error en el texto
InnerError	Error	O	Un error interno que se reportó por el

			subcomponente del componente para informes.

Implementación

En la siguiente sección se da un ejemplo sobre los recursos del registro que discutimos anteriormente. Esto es (de nuevo) solo un ejemplo.

Para que pueda comprenderlo mejor, agrupé la fuente de una forma artificial.

> Todo lo que pertenece a la API se comprime dentro de la clase del mapa de los recursos.
> El bloque completo de UrlMap XData se divide en una sola entrada Route.
> Cada entrada está "pegada" al método estático que realmente implementa la funcionalidad.

Entonces, para recuperar la clase verdadera se necesita algo de (re)ingeniería. ¡Disfrute de la (re)ingeniería!

El primer método del servicio es la consulta...

```
<!-- Query the registry. The URL query part holds select criteria. -->

<Route Url="/:service" Method="GET" Call="QueryRegistry"/>

classmethod QueryRegistry(service) as %Status {

	try {

		set serviceInstance = ..getServiceInstance(service)

		do
 ..dumpResponse(serviceInstance.runQuery(..getQueryParameters($listbuild("name","dob"
,"motherMaidenName","birthPlaceCountry","birthPlaceCity"))))

	}

	catch ex {

		do ..ReportHttpStatusCode(..getHTTPStatusCode(ex),ex.AsStatus())

	}

	quit $$$OK

}



<!-- Get a single entry -->

<Route Url="/:service/:registryID" Method="GET" Call="GetEntry"/>
```

```
classmethod GetEntry(service,registryID) as %Status {
        try {
                set serviceInstance = ..getServiceInstance(service)
                do ..dumpResponse(serviceInstance.get(registryID))
        }
        catch ex {
                do ..ReportHttpStatusCode(..getHTTPStatusCode(ex),ex.AsStatus())
        }
        quit $$$OK
}

<!-- Delete a single entry -->
<Route Url="/:service/:registryID" Method="DELETE" Call="DeleteEntry"/>
classmethod DeleteEntry(service,registryID) as %Status {
        try {
                set serviceInstance = ..getServiceInstance(service)
                do ..dumpResponse(serviceInstance.delete(registryID))
        }
        catch ex {
                do ..ReportHttpStatusCode(..getHTTPStatusCode(ex),ex.AsStatus())
        }
        quit $$$OK
}

<!-- Utility method to initialize the registry. -->
<Route Url="/:service/_init" Method="POST" Call="InitializeRegistry"/>
classmethod InitializeRegistry(service) as %Status {
        try {
```

```
                set serviceInstance = ..getServiceInstance(service)
                do ..dumpResponse(serviceInstance.init())
        }
        catch ex {
                do ..ReportHttpStatusCode(..getHTTPStatusCode(ex),ex.AsStatus())
        }
        quit $$$OK
}


<!-- Utility method to populate test data. -->
<Route Url="/:service/_populate/:numberOfRecords" Method="POST" Call="Populate"/>
classmethod Populate(service,numberOfRecords) as %Status {
        try {
                set serviceInstance = ..getServiceInstance(service)
                do ..dumpResponse(serviceInstance.populate(numberOfRecords))
        }
        catch ex {
                do ..ReportHttpStatusCode(..getHTTPStatusCode(ex),ex.AsStatus())
        }
        quit $$$OK
}


<!-- Update individual attributes of a registry entry. -->
<Route Url="/:service/:registryID" Method="POST" Call="UpdateAttribute"/>
classmethod UpdateAttribute(service,registryID) as %Status {
        try {
                set serviceInstance = ..getServiceInstance(service)
                do
 ..dumpResponse(serviceInstance.updateAttribute(registryID, ..getFormParameters($list
build("name","dob","motherMaidenName","birthPlaceCountry","birthPlaceCity"))))
```

```
        }
        catch ex {
                do ..ReportHttpStatusCode(..getHTTPStatusCode(ex),ex.AsStatus())
        }
        quit $$$OK
}


<!-- Add a new or update an existing registry entry. -->
<Route Url="/:service" Method="PUT" Call="AddOrUpdate"/>
classmethod AddOrUpdate(service) as %Status {
        try {
                set serviceInstance = ..getServiceInstance(service)
                do
 ..dumpResponse(serviceInstance.addOrUpdate(..getContentParameter()))
        }
        catch ex {
                do ..ReportHttpStatusCode(..getHTTPStatusCode(ex),ex.AsStatus())
        }
        quit $$$OK
}
```

Ahora, es momento de compartir con usted algunos métodos que son útiles.

```
classmethod getServiceInstance(serviceName) as Ens.BusinessService {
        set
 status = ##class(Ens.Director).CreateBusinessService(serviceName, .instance)
        throw:$$$ISERR(status) ##class(NoProduction).%New(status)
        quit instance
}
```

```
classmethod getHTTPStatusCode(ex) {

        quit $case(ex.%ClassName(1),

                                ##class(NoProduction).%ClassName(1)                  :5
03,

                                ##class(NotImplemented).%ClassName(1)        :501,


                               :500)

}



classmethod dumpResponse(responseObject) {

        if $isObject(responseObject) {

                if
 responseObject.%Ex
tends(##class(%DynamicObject).%ClassName(1)) { write responseObject.%ToJSON() }

                elseif
 responseObject.%Extends(##class(%ZEN.proxyObject).%ClassName(1)) {

                        do
 ##class(%ZEN.Auxiliary.jsonProvider).%ObjectToJSON(responseObject)

                }

                elseif responseObject.%Extends(##class(%XML.Adaptor).%ClassName(1)) {

                        do responseObject.XMLExportToString(.ret)

                        write ret

                }

                else { throw ##class(Serialization).%New() }

        }

        else {

                write responseObject

        }

}
```

```
classmethod getQueryParameters(parameterList) as %DynamicObject {

        set parameterObject = {}

        for i=1:1:$listlength(parameterList) {

                set parameterName=$listget(parameterList,i)

                set
 $property(parameterObject, parameterName) = %request.Get(parameterName)

        }

        quit parameterObject

}



classmethod getFormParameters(parameterList,queryObject) as %DynamicObject {

        if $data(queryObject) { set parameterObject = queryObject }

        else { set parameterObject = {} }

        for i=1:1:$listlength(parameterList) {

                set parameterName=$listget(parameterList,i)

                set
 $property(parameterObject, parameterName) = %request.Get(parameterName)

        }

        quit parameterObject

}



classmethod getContentParameter() as %DynamicObject {

        quit {}.%FromJSON(%request.Content)

}
```

Hasta aquí todo. Terminamos de diseñar (?), implementar y documentar los servicios web de una API RESTful.

#Principiante #API REST #Caché #Ensemble

10 1 0 0 245

Log in or sign up to continue

Añade la respuesta

URL de fuente: https://es.community.intersystems.com/post/api-restful