Introducción a Restful

Manuel Molino Milla

6 de noviembre de 2017

Índice

1. Introducción

1.1. ¿Historia de REST?

- Es una arquitectura de software que surgió en 1999
- Los principios que se basa REST son:
  1. Todo es un recurso.
  2. Cada recurso es identificado por un único identificador (URI).
  3. Usa los métodos de HTTP
  4. Los recursos pueden tener múltiples representaciones.
  5. Es una comunicación sin estado.

1.1.1. Todo es un recurso

- Para entender esta idea, debemos entender que la representación de datos no como un fichero físico sino como un formato específico.
- Usamos el *content-type* para su descripción.
- Ejemplos:
  1. image/jpeg
  2. video/mpeg
  3. text/html
  4. ...

1.1.2. Cada recurso es identificado como una URI

- Sobre todo en Internete cada recurso debe ser único y accesible según su URI.
- Además este formato debe tener un formato amigable (fácil de entender)
- Ejemplo de URI:
  1. http://www.mydatastore.com/images/vacation/2014/summer
  2. http://www.mydatastore.com/videos/vacation/2014/winter
  3. http://www.mydatastore.com/data/documents/balance?format=xml
  4. http://www.mydatastore.com/data/archives/2014

1.1.3. Uso de los métodos estándar de HTTP

- GET
- POST
- PUT
- DELETE
- HEAD
- OPTIONS
- TRACE
- CONNECT

Existe similitud con SQL con acciones de tipo CRUD relacionados con las sentencias *INSERT, SELECT, UPDATE y DELETE*
Para aplicar correctamente los principios de REST los verbos de HTTP deberían ser usado como sigue:

HTTP verb	**Action**	**Response status code**
GET	Request an existing resource	"**200 OK**" if the resource exists, "**404 Not Found**" if it does not exist, and "**500 Internal Server Error**" for other errors
PUT	Create or update a resource	"**201 CREATED**" if a new resource is created, "**200 OK**" if updated, and "**500 Internal Server Error**" for other errors
POST	Update an existing resource	"**200 OK**" if the resource has been updated successfully, "**404 Not Found**" if the resource to be updated does not exist, and "**500 Internal Server Error**" for other errors
DELETE	Delete a resource	"**200 OK**" if the resource has been deleted successfully, "**404 Not Found**" if the resource to be deleted does not exist, and "**500 Internal Server Error**" for other errors

Generalmente nos encontramos como el método POST usado para la creación de recursos, pero en el caso que se crea con una específica URI debe usarse el método PUT:

```
PUT /data/documents/balance/22082014 HTTP/1.1
Content-Type: text/xml
Host: www.mydatastore.com
<?xml version="1.0" encoding="utf-8"?>
<balance date="22082014">
<Item>Sample item</Item>
<price currency="EUR">100</price>
</balance>
HTTP/1.1 201 Created
Content-Type: text/xml
Location: /data/documents/balance/22082014
```

En el caso que sea la aplicación la que decide donde colocar dicho recurso, se usa el método POST:

```
POST /data/documents/balance HTTP/1.1
Content-Type: text/xml
Host: www.mydatastore.com
<?xml version="1.0" encoding="utf-8"?>
<balance date="22082014">
<Item>Sample item</Item>
<price currency="EUR">100</price>
</balance>
HTTP/1.1 201 Created
Content-Type: text/xml
Location: /data/documents/balance
```

1.1.4. Los recursos pueden tener múltiples representaciones

La representación puede ser variada, por ejemplo usando formato xml o json.

```
POST /data/documents/balance HTTP/1.1
Content-Type: application/json
Host: www.mydatastore.com
{
"balance": {
"date": ""22082014"",
"Item": "Sample item",
"price": {
       "-currency": "EUR",
       "#text": "100"
       }
  }
}
HTTP/1.1 201 Created
Content-Type: application/json
Location: /data/documents/balance
```

1.1.5. Comunicación sin estado

- Todas las operaciones llevadas acabo dentro de una petición HTTP deben ser átomicas.
- Todas las modificaciones de un recurso deben llevarse a cabo dentro de la misma petición.
- Después de la petición HTTP, el recurso queda en un estado final.
- Esto implica que actualizaciones parciales del recurso no se permiten.

1.1.6. Confiabilidad

Los métodos HTTP pueden ser seguros e idempotentes

Seguros los métodos HTTP son seguros si las peticiones HTTP no afectan al estado de los recursos, es decir no hay modificaciones ni efectos laterales.

Idempotentes Cuando la respuesta es siempre la misma, independientemente de las veces que lo llevemos acabo.

HTTP Method	Safe	Idempotent
GET	Yes	Yes
POST	No	No
PUT	No	Yes
DELETE	No	Yes

2. Ejemplos de aplicaciones REST

Ejemplo 1:

```
Some sample RESTful API endpoint URLs are as follows:
• GET http://myapi.com/v1/accounts : This returns a list of accounts
• GET http://myapi.com/v1/accounts/1 : This returns a single account
by Id: 1
• POST http://myapi.com/v1/accounts : This creates a new account
(data submitted as a part of the request)
• PUT http://myapi.com/v1/accounts/1 : This updates an existing
account by Id: 1 (data submitted as part of the request)
• GET http://myapi.com/v1/accounts/1/orders : This returns a list of
orders for account Id: 1
• GET http://myapi.com/v1/accounts/1/orders/21345 : This returns the
details for a single order by Order Id: 21345 for account Id: 1
```

Ejemplo 2:

```
GET /api/attractions
Retrieves attractions. Takes lat , lng , and radius as querystring parameters and
returns a list of attractions.
GET /api/attraction/:id
Returns an attraction by ID.
POST /api/attraction
Takes lat , lng , name , description , and email in the request body. The newly added
attraction goes into an approval queue.
PUT /api/attraction/:id
Updates an existing attraction. Takes an attraction ID, lat , lng , name , descrip
tion , and email . Update goes into approval queue.
DEL /api/attraction/:id
Deletes an attraction. Takes an attraction ID, email , and reason . Delete goes into
approval queue.
```

Ejemplo 3:

URL	HTTP Verb	POST Body	Result
/api/movies	GET	empty	Returns all movies
/api/movies	POST	JSON String	New movie Created
/api/movies/:id	GET	empty	Returns single movie
/api/movies/:id	PUT	JSON string	Updates an existing movie
/api/movies/:id	DELETE	empty	Deletes existing movie

Ejemplo 4:

Resource	POST create	GET read	PUT update	DELETE delete
/dogs	create a new dog	list dogs	bulk update dogs	delete all dogs
/dogs/1234	error	show Bo	if exists update Bo if not error	delete Bo

Ejemplo 5:

RESTful API HTTP methods

Resource	GET	PUT	POST	DELETE
Collection URI, such as `http://api.example.com/resources/`	**List** the URIs and perhaps other details of the collection's members.	**Replace** the entire collection with another collection.	**Create** a new entry in the collection. The new entry's URI is assigned automatically and is usually returned by the operation.[10]	**Delete** the entire collection.
Element URI, such as `http://api.example.com/resources/item17`	**Retrieve** a representation of the addressed member of the collection, expressed in an appropriate Internet media type.	**Replace** the addressed member of the collection, or if it does not exist, **create** it.	Not generally used. Treat the addressed member as a collection in its own right and **create** a new entry in it.[10]	**Delete** the addressed member of the collection.

Ejemplo 6:

Method	URL	Meaning
GET	`/messages.json`	Return list of messages in JSON format.
PUT	`/messages.json`	Update/replace all messages and return status/error in JSON.
POST	`/messages.json`	Create new message and return its ID in JSON format.
GET	`/messages/{id}.json`	Return message with ID `{id}` in JSON format.
PUT	`/messages/{id}.json`	Update/replace message with ID that equals the value of `{id}`; if `{id}` message doesn't exist, create it.
DELETE	`/messages/{id}.json`	Delete message with ID `{id}`, and return status/error in JSON format.

Ejemplo 7:

Table 2-1. *List of Endpoints and How They Change When the REST Style Is Applied*

Old Style	REST Style
/getAllBooks	GET /books
/submitNewBook	POST /books
/updateAuthor	PUT /authors/:id
/getBooksAuthors	GET /books/:id/authors
/getNumberOfBooksOnStock	GET /books (This number can easily be returned as part of this endpoint.)
/addNewImageToBook	PUT /books/:id
/getBooksImages	GET /books/:id/images
/addCoverImage	POST /books/:id/cover_image
/listBooksCovers	GET /books (This information can be returned in this endpoint using subresources.)

3. Ejercicio

Crea la estructura de una aplicación rest para la gestión de usuarios. Los usuarios se identifican por un *id* único y tienen como propiedades nombre, password y profesión.
La aplicación debe permitir las siguientes acciones:

- Listar todos los usuarios.
- Listar un usuario determinado por loa *id*
- Borrar un usuario.
- Añadir un usuario.
- Actualizar un usuario.

Rellena una tabla con el siguiente encabezado:

URL	método HTTP	POST body	Resultado