Sistemas Orientados a Servicios

Diseño e implementación de un servicio web RESTful

Miembros

- *Víctor Nieves Sánchez*
- *Daniel Morgera Pérez*

Fecha

22 de abril de 2019

Índice

1 Introducción

Representational State Transfer o más conocido como *REST* un estilo de arquitectura *software* para sistemas hipermedia distribuidos como la *World Wide Web*. [1]

RESTful hace referencia a un servicio web que implementa la arquitectura *REST*.

La arquitectura *REST* fue definida por *Roy Fielding* en el año 2000, que además es unos de los principales artífices de la especificación del protocolo *HTTP*.

La principal ventaja de esta arquitectura es que ha aportado a la web una mayor escalabilidad, es decir, dan soporte a un mayor número de componentes y las interacciones entre ellos. Esta ventaja es gracias a una serie de características que presenta la arquitectura *REST*: [2] [3]

- Protocolo cliente/servidor sin estado
- 4 operaciones básicas: *GET, POST, PUT* y *DELETE*
- Los objetos se manipulan a partir de la *URI* (*Uniform Resource Identifier*)
- Interfáz uniforme
- Sistema de capas
- Uso de hipermedios

Algunas de las ventajas que ofrece *REST* para el desarrollo son: [4]

- Separación entre el cliente y el servidor
- Visibilidad, fiabilidad y escalabilidad
- Es independiente del tipo de plataformas o lenguajes

En esta práctica se propone el diseño e implementación de una *API REST* y de un prototipo funcional de un servicio sencillo para una red social de tipo *Facebook*, donde los usuarios pueden interactuar entre ellos de diversas formas.

También se diseñará un cliente interactivo con el cual poder realizar todas las operaciones de nuestra *API REST* a través de un terminal. Pudiendo elegir todos los parámetros y filtros que estén implementados en la *API REST*.

El lenguaje empleado en este trabajo ha sido *JAVA*.

2 Aclaraciones y ambiguedades

1) Es **muy importante** que se escriban bien las *URI* para que todo funcione correctamente. Las *URI* **no pueden terminar en "/"**, por ejemplo *http://localhost:8080/upmsocial/usuarios/* NO sería correcta, ya que debería ser *http://localhost:8080/upmsocial/usuarios* (sin la "/").

2) En nuestro proyecto se podrá crear un nuevo usuario siempre y cuando no esté el email ya asignado a algún otro usuario. El resto de valores (salvo ID e email) si pueden ser idénticos.

3) No será necesario que dos amigos sean amigos para poder enviarse mensajes privados entre sí.

3 Recursos y operaciones

Nuestra URI global será: ***upmsocial.com/es***

Se han identificado los siguientes recursos y operaciones:

3.1 Usuarios

URI	upmsocial.com/es/usuarios
Descripción	Lista los usuarios del sistema
Método	**GET**
Cadena de consulta	filterBy=name
Cuerpo	-
Devuelve	200 OK 404 Not Found 500 Internal Server Error

URI	upmsocial.com/es/usuarios
Descripción	Crea un nuevo usuario
Método	**POST**
Cadena de consulta	-
Cuerpo	usuarios/id_usuario + JSON
Devuelve	201 Created 406 Not Acceptable 500 Internal Server Error

3.2 Usuario (concreto)

URI	upmsocial.com/es/usuarios/id_usuario
Descripción	Obtiene la información de un usuario concreto
Método	**GET**
Cadena de consulta	-
Cuerpo	-
Devuelve	200 OK 404 Not Found 500 Internal Server Error

URI	upmsocial.com/es/usuarios/id_usuario
Descripción	Modifica la información de un usuario concreto
Método	**PUT**
Cadena de consulta	-
Cuerpo	usuarios/id_usuario + JSON
Devuelve	200 OK 404 Not Found 406 Not Acceptable 500 Internal Server Error

URI	upmsocial.com/es/usuarios/id_usuario
Descripción	Elimina a un usuario
Método	**DELETE**
Cadena de consulta	-
Cuerpo	-
Devuelve	200 OK 404 Not Found 500 Internal Server Error

3.3 Amigos

URI	upmsocial.com/es/usuarios/id_usuario/amigos
Descripción	Lista los amigos de un usuario
Método	**GET**
Cadena de consulta	filterBy=name start=number end=number
Cuerpo	-
Devuelve	200 OK 404 Not Found 500 Internal Server Error

URI	upmsocial.com/es/usuarios/id_usuario/amigos
Descripción	Añade un amigo al usuario
Método	**POST**
Cadena de consulta	-
Cuerpo	amigos/id_usuario + JSON
Devuelve	201 Created 404 Not Found 406 Not Acceptable 500 Internal Server Error

3.4 Amigo (concreto)

URI	upmsocial.com/es/usuarios/id_usuario/amigos/ id_amigo
Descripción	Elimina una relación de amistad
Método	**DELETE**
Cadena de consulta	-
Cuerpo	-
Devuelve	200 OK 404 Not Found 500 Internal Server Error

3.5 Muro personal

URI	upmsocial.com/es/usuarios/id_usuario/ muro_personal
Descripción	Obtiene una lista de los mensajes de un usuario en su muro personal
Método	**GET**
Cadena de consulta	filterBy=name start=number end=number
Cuerpo	-
Devuelve	200 OK 404 Not Found 500 Internal Server Error

URI	upmsocial.com/es/usuarios/id_usuario/ muro_personal
Descripción	Publica un mensaje en el muro personal
Método	**POST**
Cadena de consulta	-
Cuerpo	muro_personal/id_mensaje + JSON
Devuelve	201 Created 404 Not Found 500 Internal Server Error

3.6 Mensaje (concreto) en muro personal

URI	upmsocial.com/es/usuarios/id_usuario/ muro_personal/id_mensaje
Descripción	Muestra un mensaje concreto del muro personal del usuario
Método	**GET**
Cadena de consulta	-
Cuerpo	-
Devuelve	200 OK 404 Not Found 500 Internal Server Error

URI	upmsocial.com/es/usuarios/id_usuario/ muro_personal/id_mensaje
Descripción	Modifica un mensaje concreto del muro personal
Método	**PUT**
Cadena de consulta	-
Cuerpo	muro_personal/id_mensaje + JSON
Devuelve	200 OK 404 Not Found

URI	upmsocial.com/es/usuarios/id_usuario/ muro_personal/id_mensaje
Descripción	Elimina un mensaje del muro personal
Método	**DELETE**
Cadena de consulta	-
Cuerpo	-
Devuelve	200 OK 404 Not Found 500 Internal Server Error

3.7 Mensajes privados

URI	upmsocial.com/es/usuarios/id_usuario/mensajes
Descripción	Obtiene una lista de los mensajes de un usuario en sus mensajes privados
Método	**GET**
Cadena de consulta	filterBy=friend_name
Cuerpo	-
Devuelve	200 OK 404 Not Found 500 Internal Server Error

URI	upmsocial.com/es/usuarios/id_usuario/mensajes
Descripción	Crea un mensaje privado
Método	**POST**
Cadena de consulta	-
Cuerpo	mensajes/id_mensaje_p + JSON
Devuelve	201 Created 404 Not Found 406 Not Acceptable 500 Internal Server Error

3.8 Mensaje (concreto) en mensajes privados

URI	upmsocial.com/es/usuarios/id_usuario/mensajes/id_mensaje_p
Descripción	Muestra un mensaje privado concreto
Método	**GET**
Cadena de consulta	-
Cuerpo	-
Devuelve	200 OK 404 Not Found 500 Internal Server Error

3.9 Muro de amigos

URI	upmsocial.com/es/usuarios/id_usuario/muro_amigos
Descripción	Obtiene una lista de publicaciones de todos los amigos de un usuario, ordenados por fecha
Método	**GET**
Cadena de consulta	filterBy=pattern
Cuerpo	-
Devuelve	200 OK 404 Not Found 500 Internal Server Error

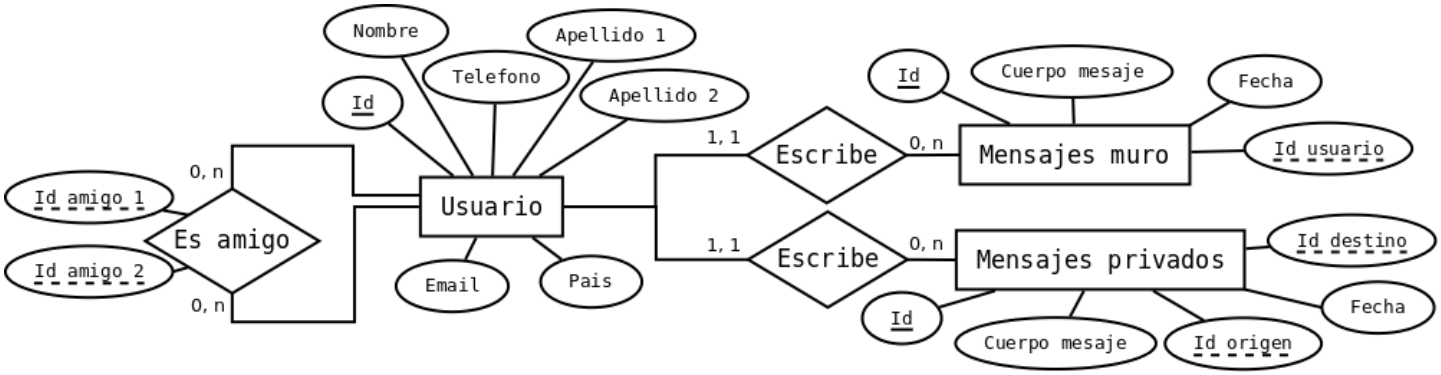
3.10 Información para el movil de un usuario

URI	upmsocial.com/es/usuarios/id_usuario/info_movil
Descripción	Muestra los datos básicos de un usuario, su último mensaje en la página, el número de amigos y los 10 últimos mensajes de las páginas de sus amigos que se han actualizado.
Método	**GET**
Cadena de consulta	-
Cuerpo	-
Devuelve	200 OK 404 Not Found 500 Internal Server Error

4 Persistencia de los datos

Se ha elegido usar una base de datos para mantener la información en el sistema.

4.1 Diagrama E-R



4.2 Ejemplos de tablas

1. Tabla Usuarios

Id	Nombre	Apellido 1	Apellido 2	Email	País	Teléfono
1234	Victor	Nieves	Sanchez	victor@email.com	España	638745901
2345	Daniel	Morgera	Perez	daniel@email.com	España	698002347
3456	Gerard	Henrison	Denver	gerard@email.com	Canadá	789432112

2. Tabla de Mensajes muro

Id	Id usuario	Cuerpo mensaje	Fecha
1357	1234	Espero aprobar todas este semestre, Jajaja!	12/04/2019
2468	2345	Pff …	17/05/2019
8932	3456	I love “The Witcher” saga!	20/07/2019

3. Tabla de Mensajes privados

Id	Id origen	Id destino	Cuerpo mensaje	Fecha
2222	1234	2345	Deberíamos ponernos con SOS ya!	10/03/2019
2223	2345	1234	Estoy de acuerdo	11/13/2019
5566	3421	4520	オタク	14/11/2019

4. Tabla de relación de amistad

Id amigo 1	Id amigo 2
1234	2345
1234	3456
2345	1234
3456	1234

5 Capturas de la ejecución de las operaciones

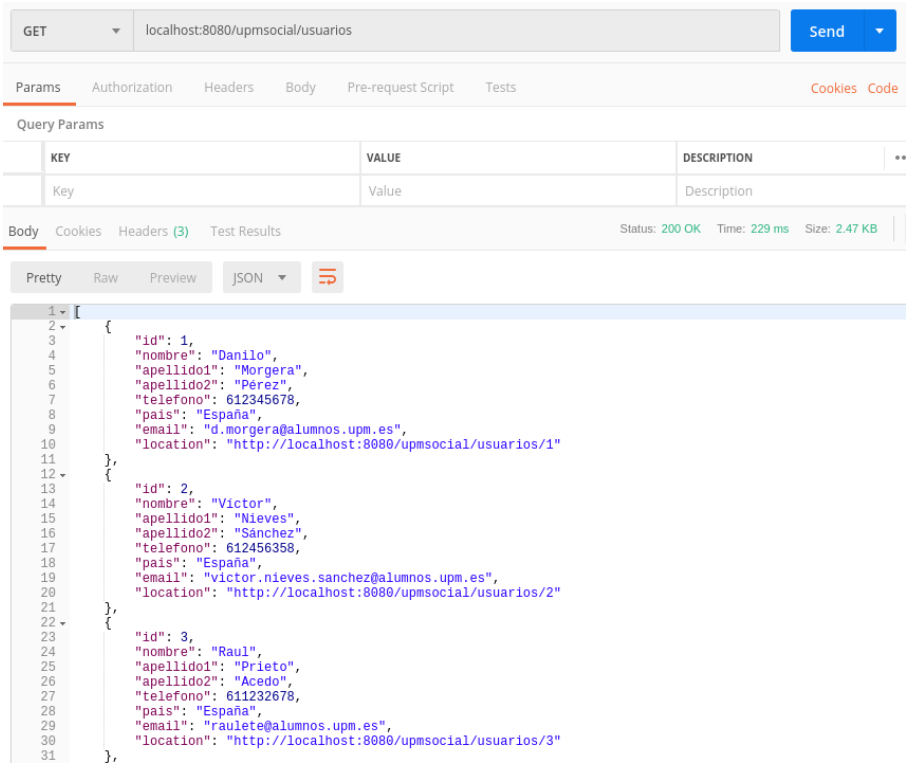
A continuación se mostrarán las capturas en *Postman* de nuestras peticiones. Se mostrarán los codigos descritos en las tablas anteriores.

Cabe indicar que no habrá imágenes del error *500 Internal Server Error* ya que estos errores solo saltarán cuando por ejemplo, la base de datos no esté disponible.

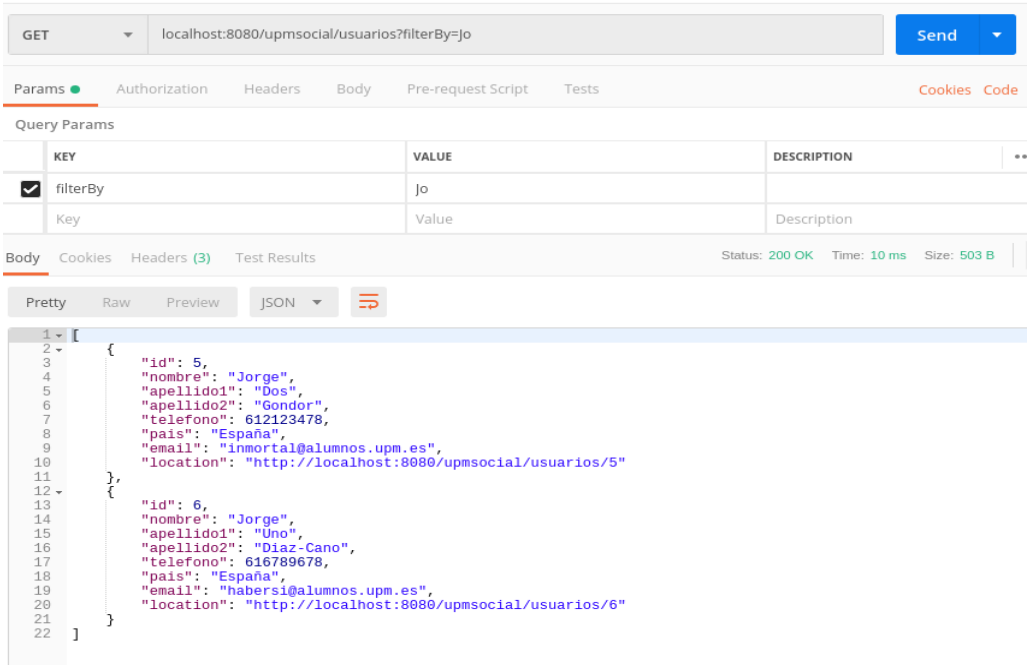
5.1 Usuarios

1. GET

200 OK

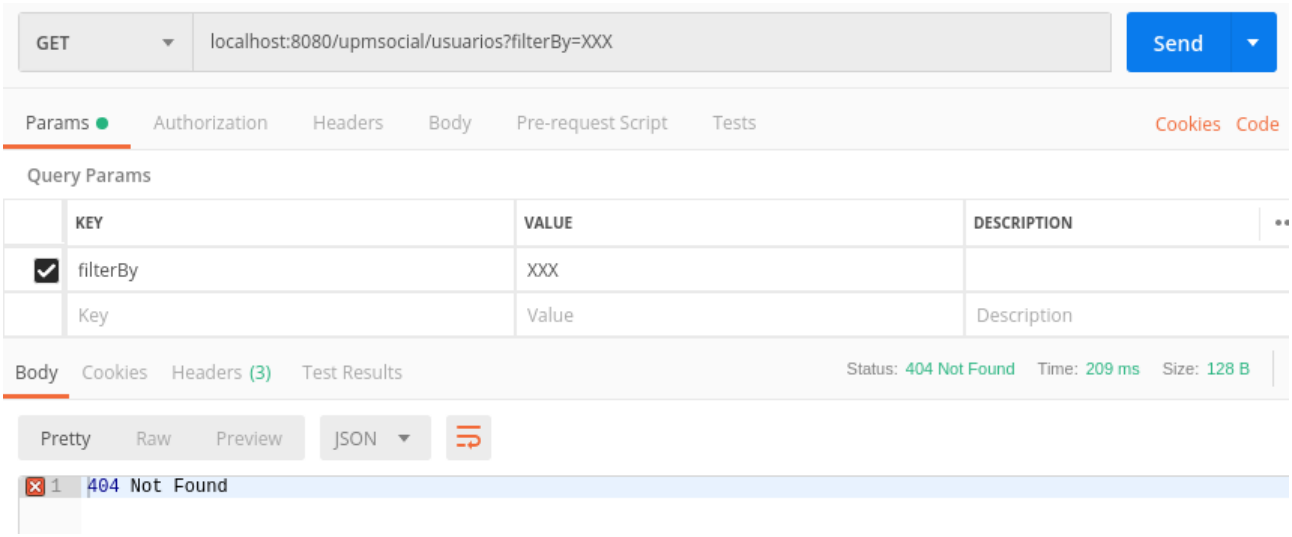


Se muestra una lista de Usuarios con todos los atributos, además de mostrar en *"location"* la *URI* navegable del recurso.



También se ha implementado un filtro "*filterBy*" el cual filtra por nombre la lista de usuarios.

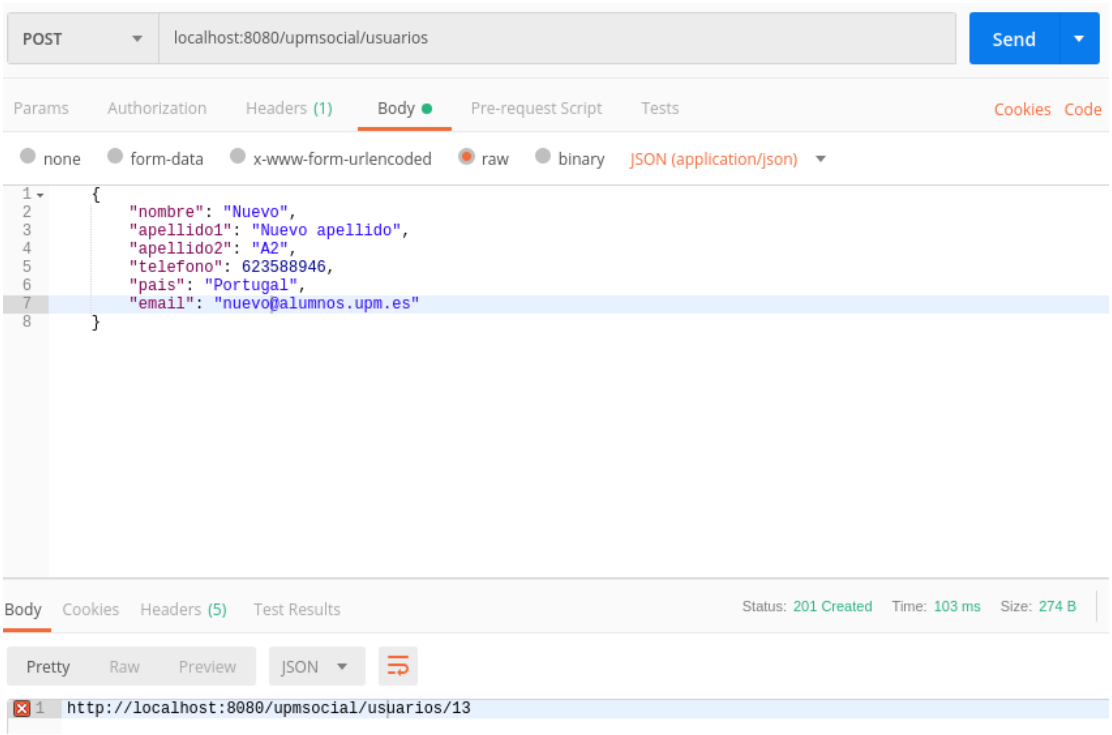
404 Not Found



Para comprobar un error *404 Not Found* se ha buscado a un usuario que en el nombre contenga "XXX", el cual no existe en nuestra base de datos.

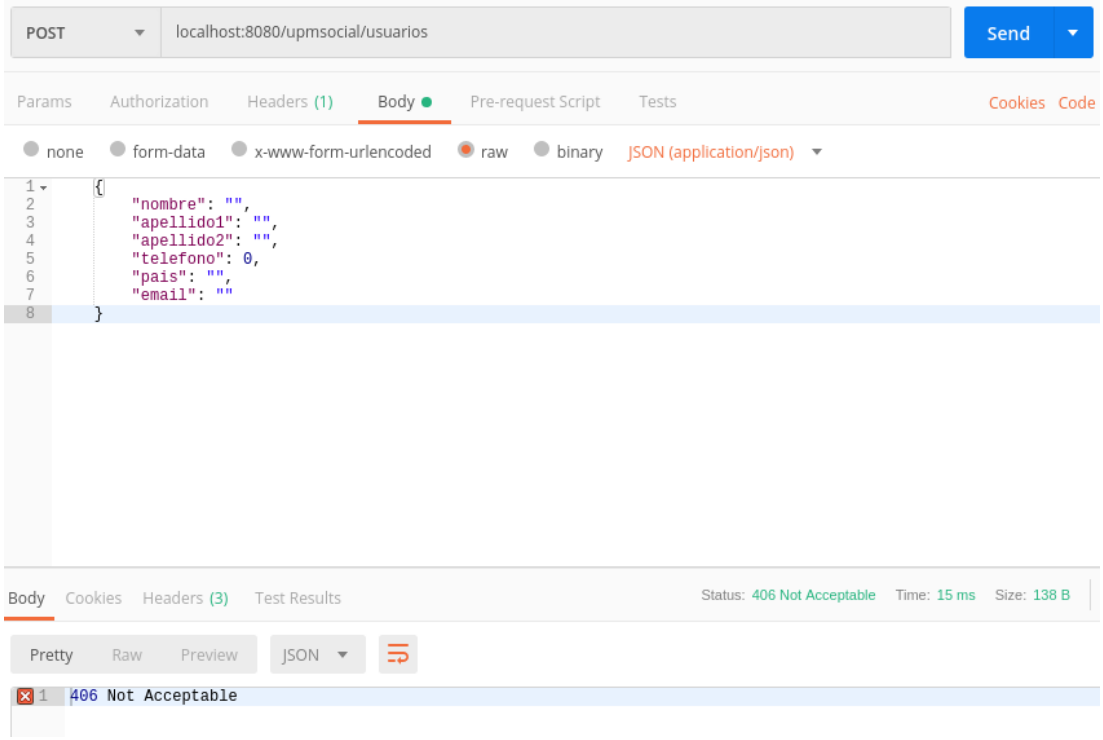
2. POST

201 Created



A la hora de crear un nuevo usuario, se devuelve solo la nueva *URI* navegable.

406 Not Acceptable

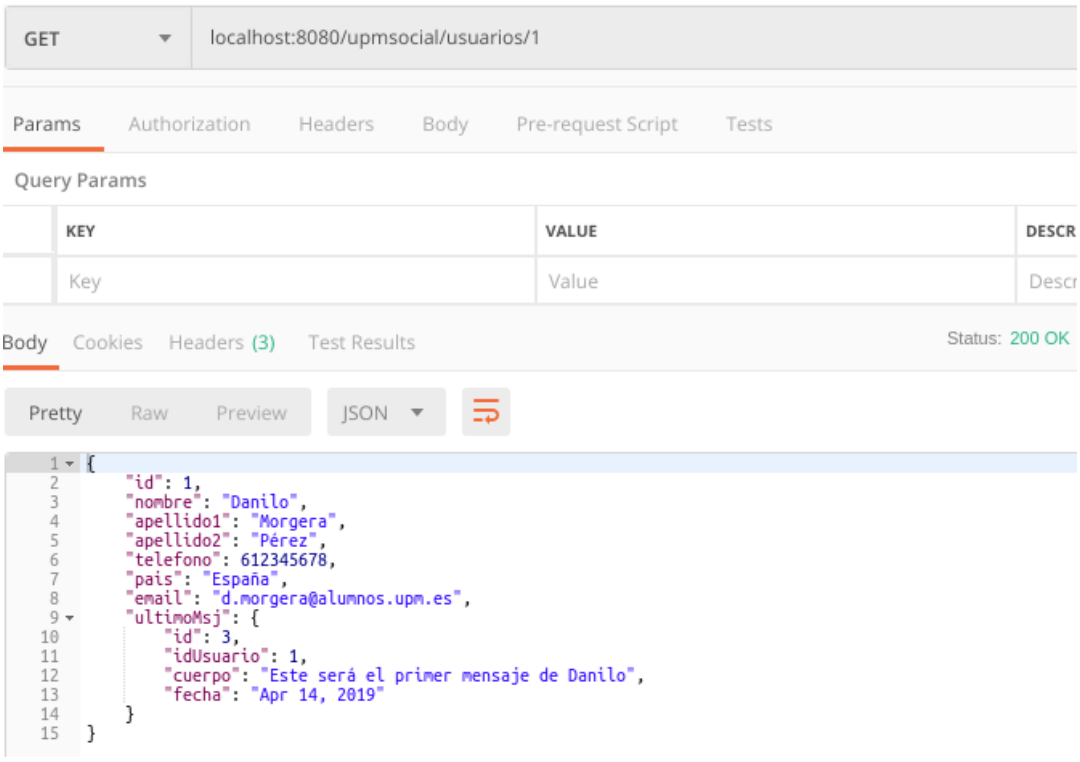


Se comprueba que los valores del *JSON* sean válidos, si no, se devuelve un error *406 Not Acceptable*.

5.2 Usuario (concreto)

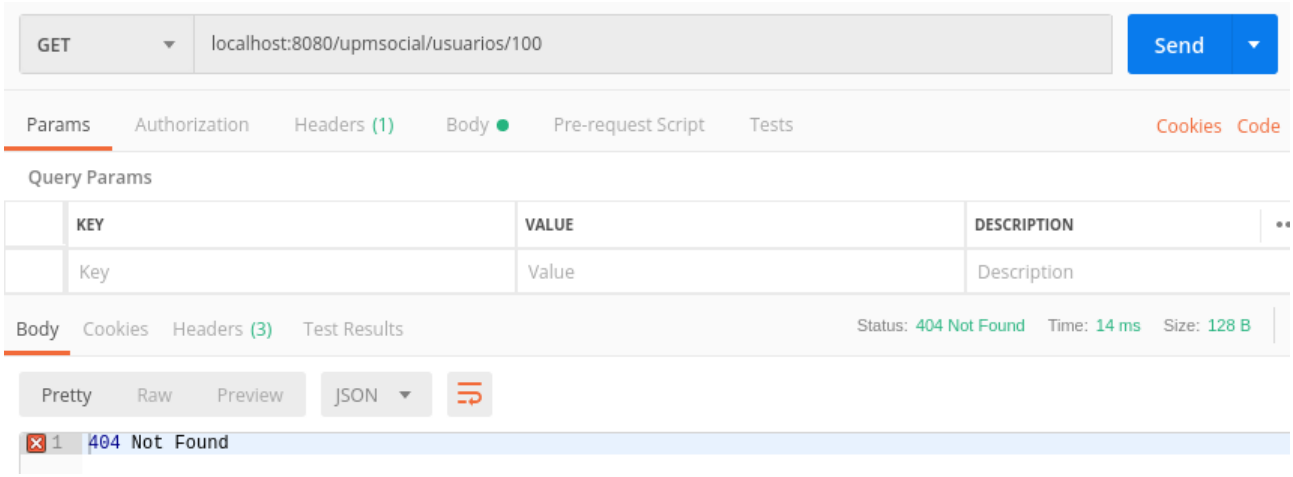
1. GET

200 OK



El get de un Usuario concreto devuelve todos sus parametros. No se ha incluido el atributo *"location"* ya que se es reduntante, al saber la propia *URI* del usuario.

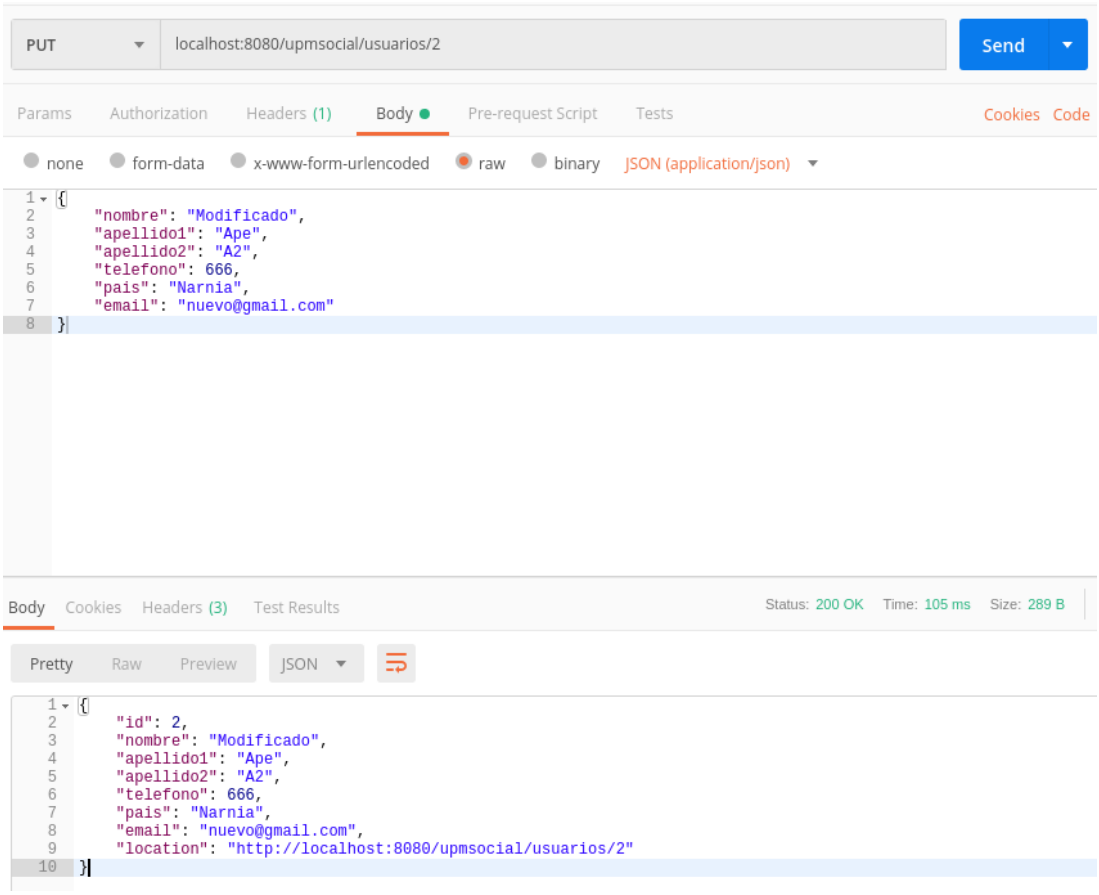
404 Not Found



Si el usuario no se encuentra en la base de datos, se devuelve el error correspondiente.

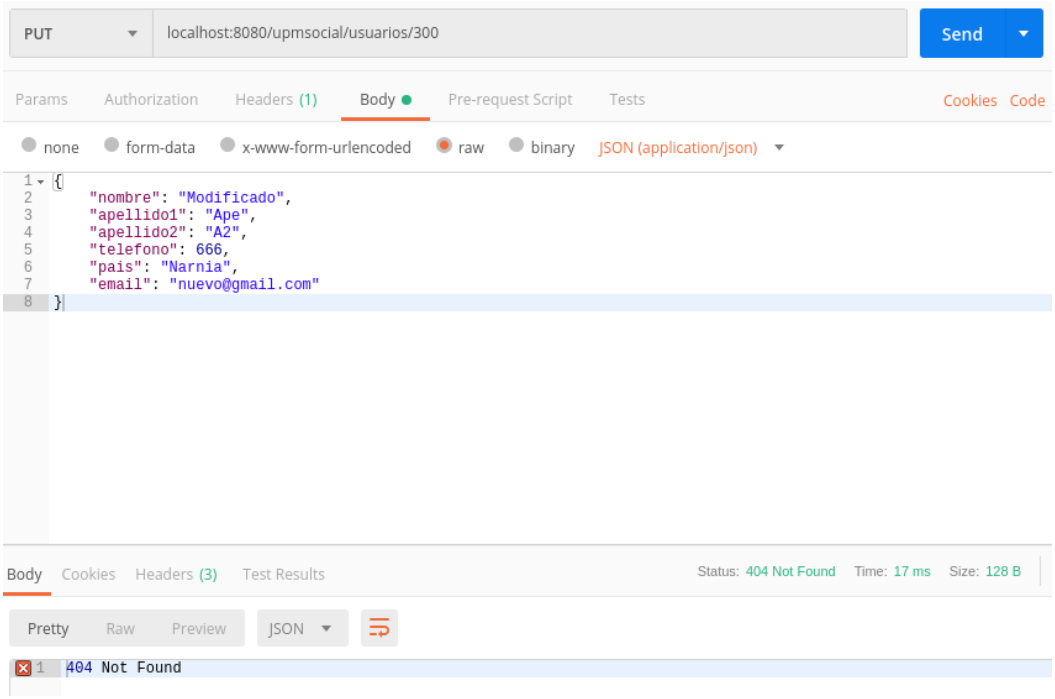
2. PUT

200 OK



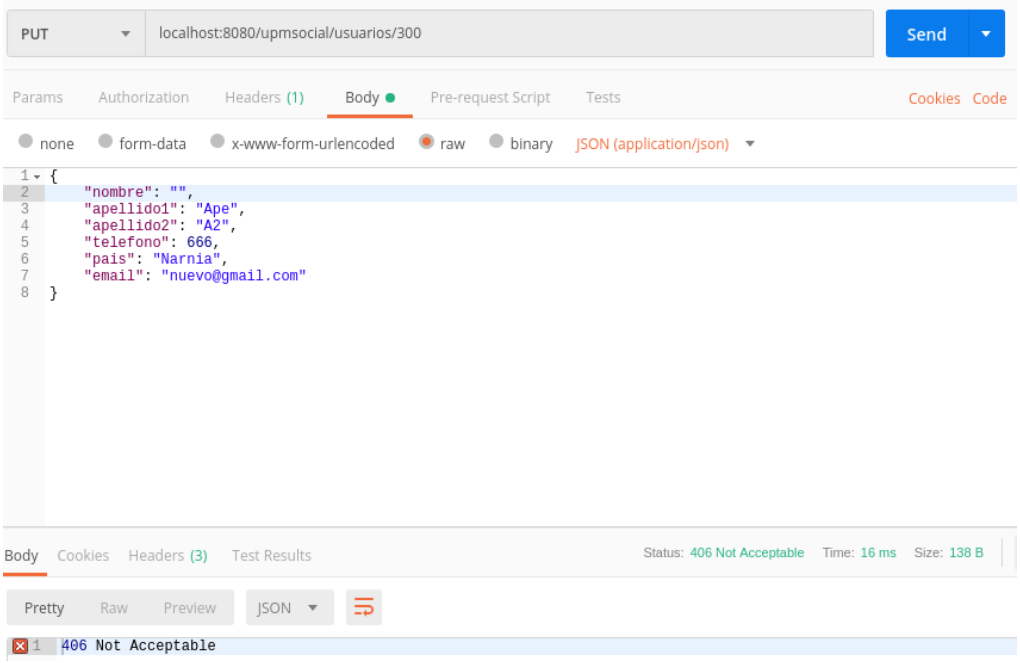
A la hora de editar un usuario existente mediante *PUT*, se pueden modificar todos sus atributos, y devuelve el *JSON* del usuario modificado además de su location (para ver que esta no cambia).

404 Not Found



Un usuario inexistente no se puede modificar (el usuario con ID 300 no existe).

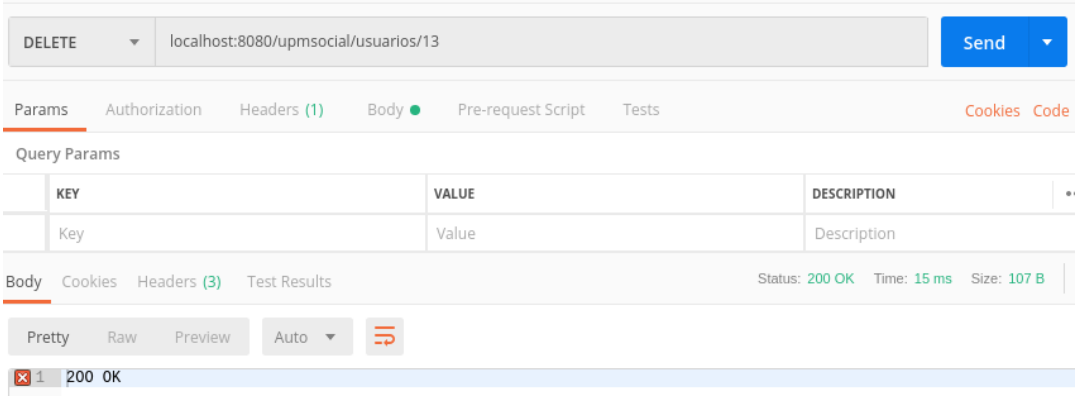
406 Not Acceptable



También se comprueban que los atributos sean válidos (nombre vacío o algún otro atributo).

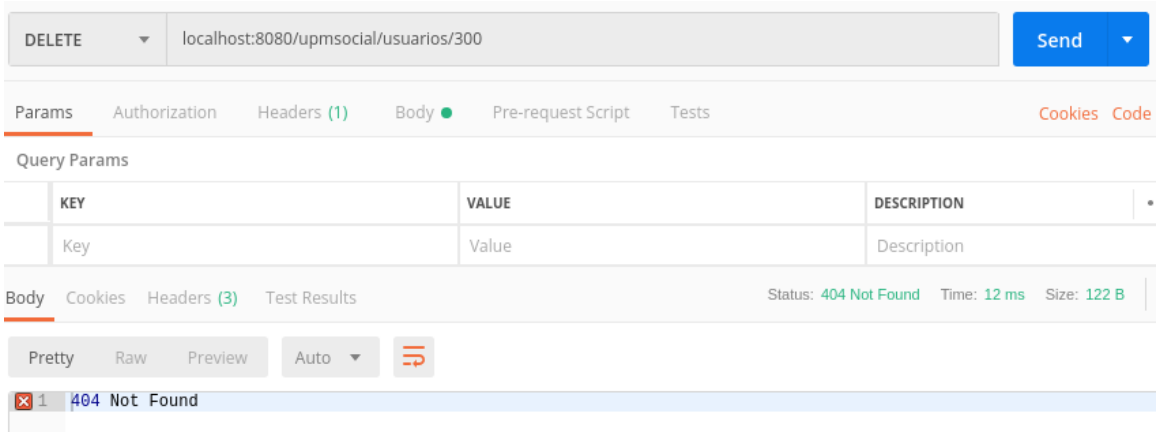
3. DELETE

200 OK



Un usuario existente, al borrarse mediante *DELETE*, se devuelve un *200 OK*.

404 Not Found

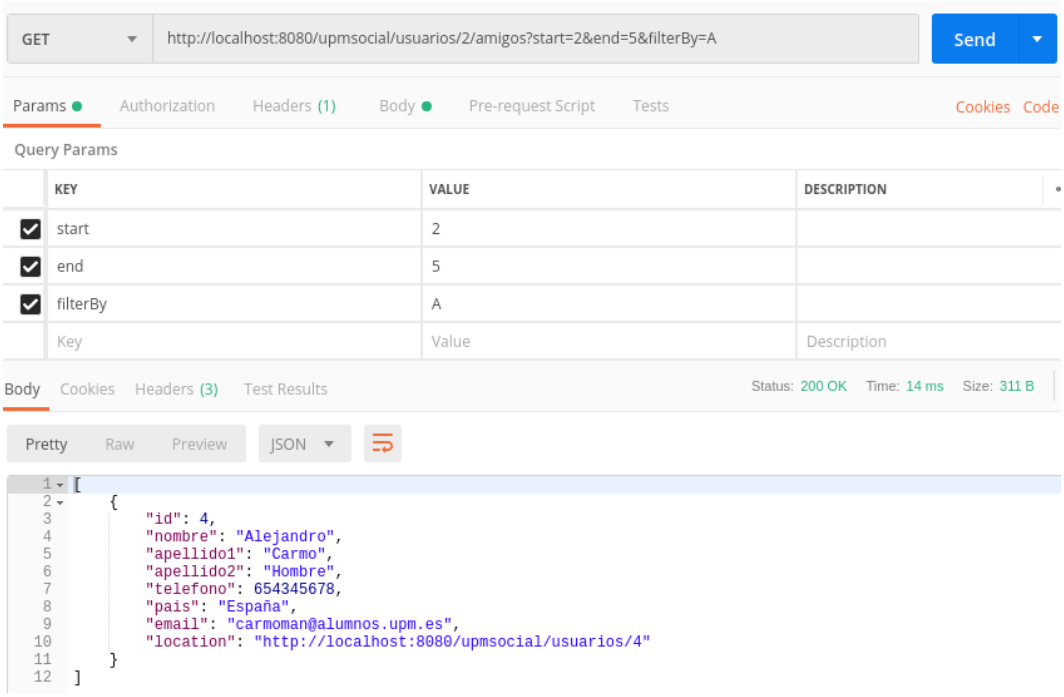


Si el usuario no existe, se devuelve de nuevo un error 404.

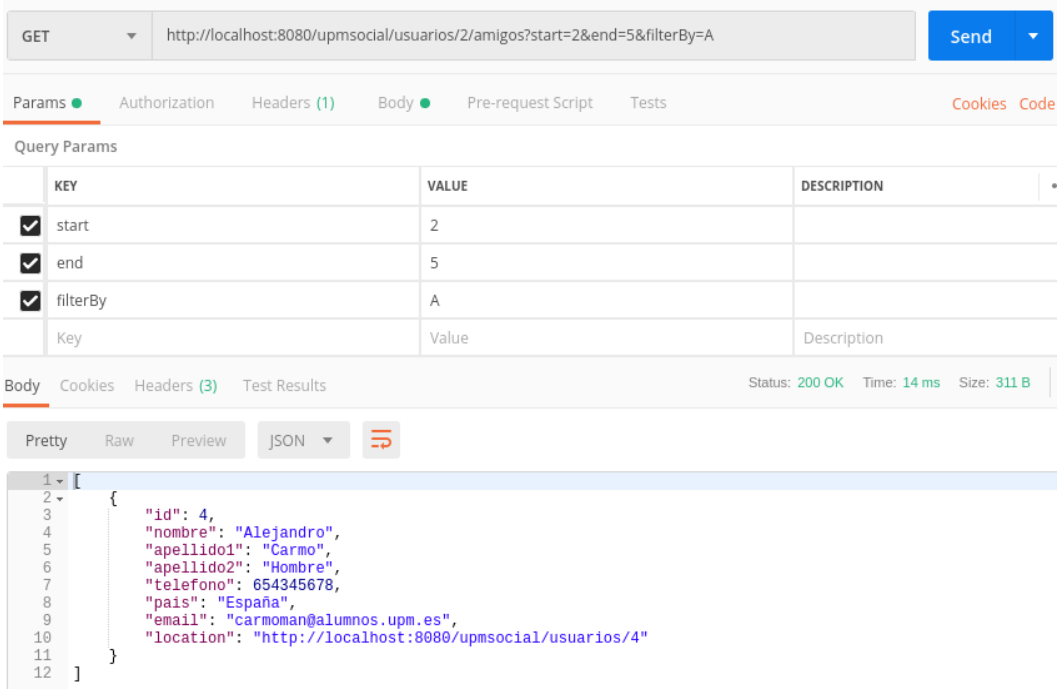
5.3 Amigos

1. GET

200 OK

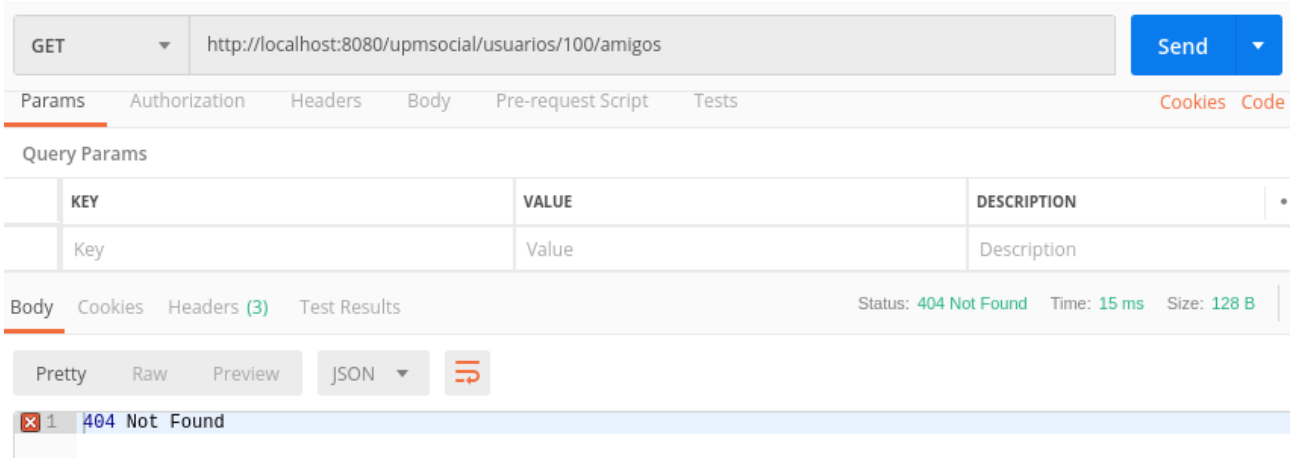


Al listar los amigos de un usuario concreto, se muestran los *JSON* de cada usuario y su *URI*.



También se ha implementado diversas maneras de filtrar el resultado

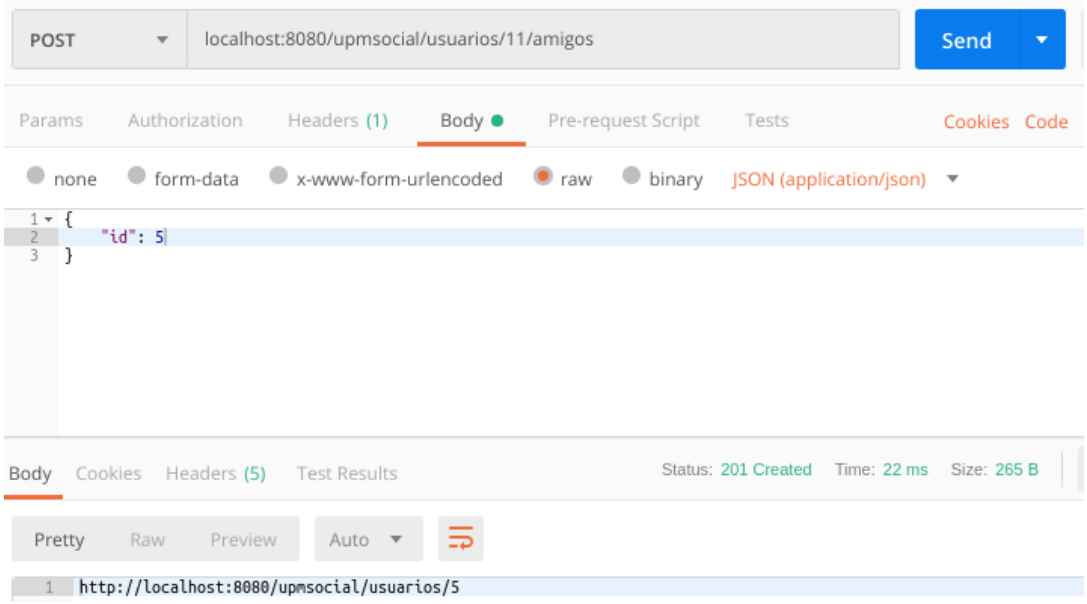
404 Not Found



Si el usuario no existe, no se podrá encontrar su lista de amigos.

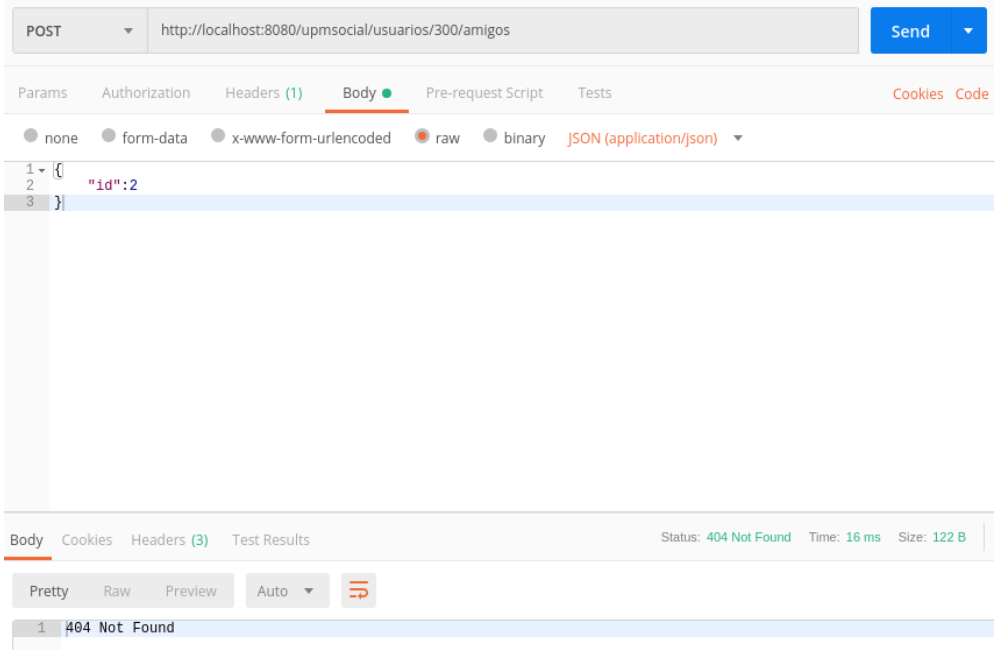
2. POST

201 Created



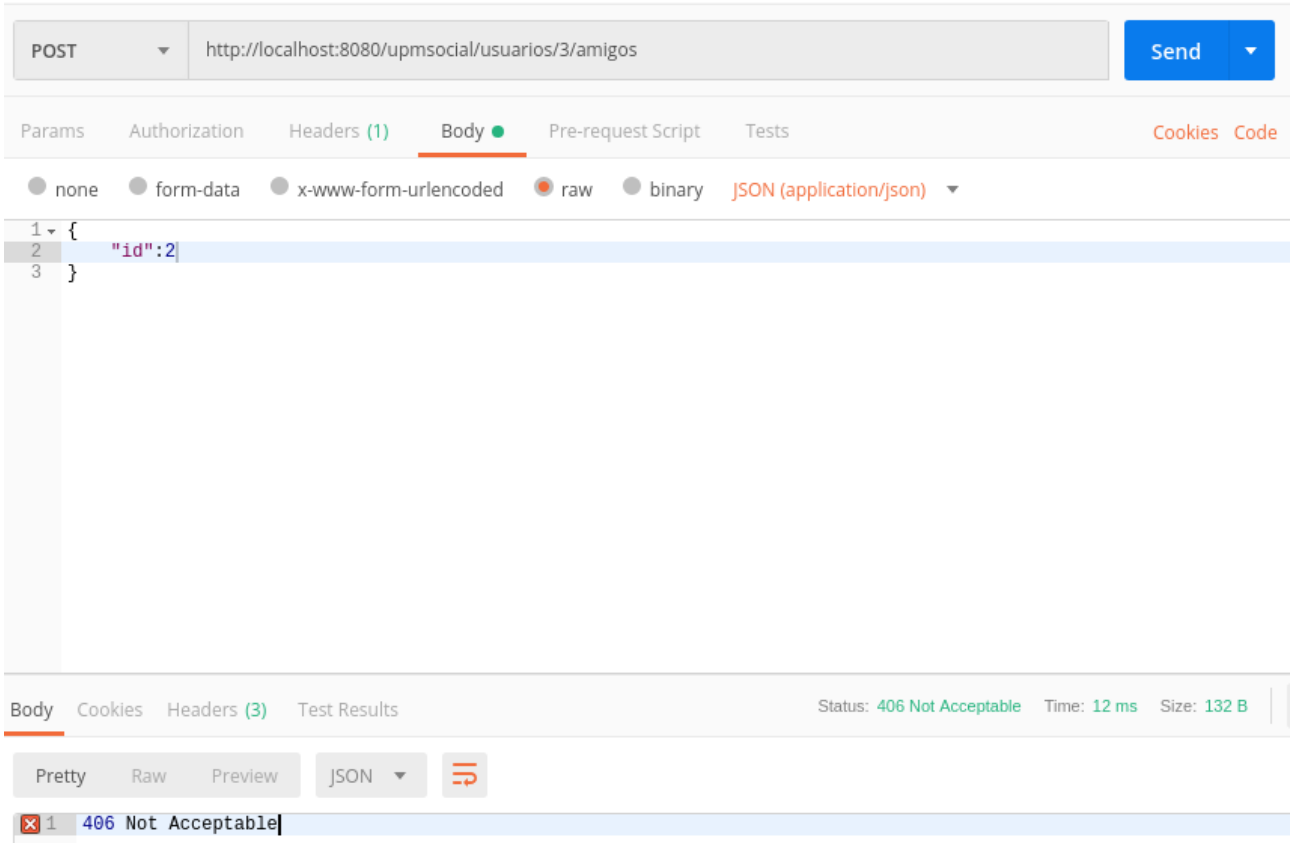
Cuando no hay una relación de amistad existente en la base de datos, y se crea una nueva, devolvemos solo la *URI* del usuario correspondiente al nuevo amigo.

404 Not Found



Si alguno de los dos usuarios no existe en el sistema, se devuelve un *404 Not Found*.

406 Not Acceptable

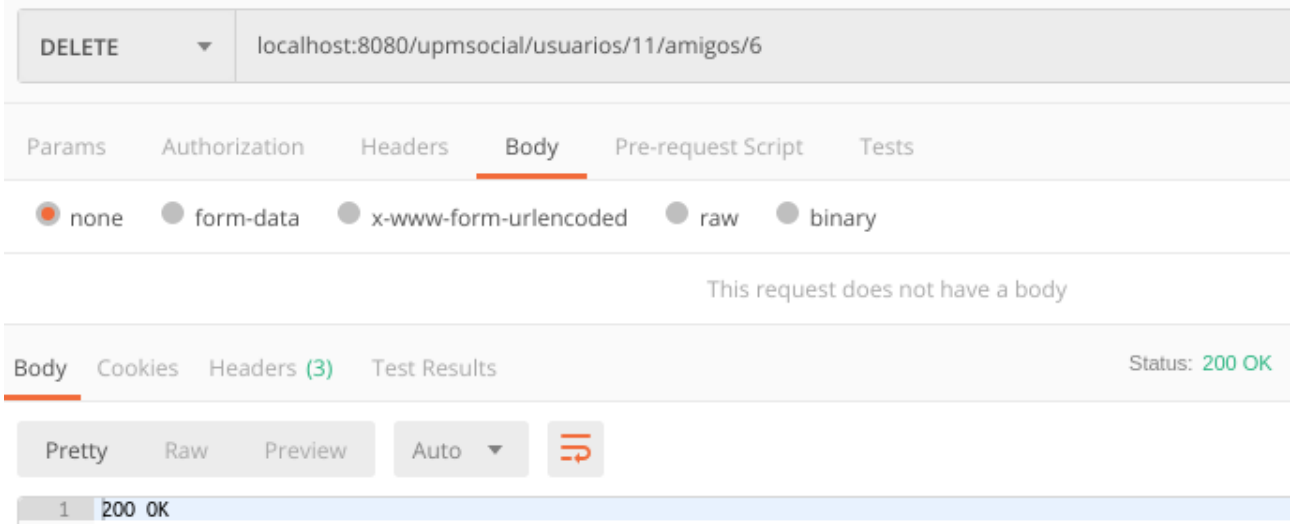


Si la relación de amistad ya existe, el *JSON* no es válido, se devuelve un *406 Not Acceptable*.

5.4 Amigo (concreto)

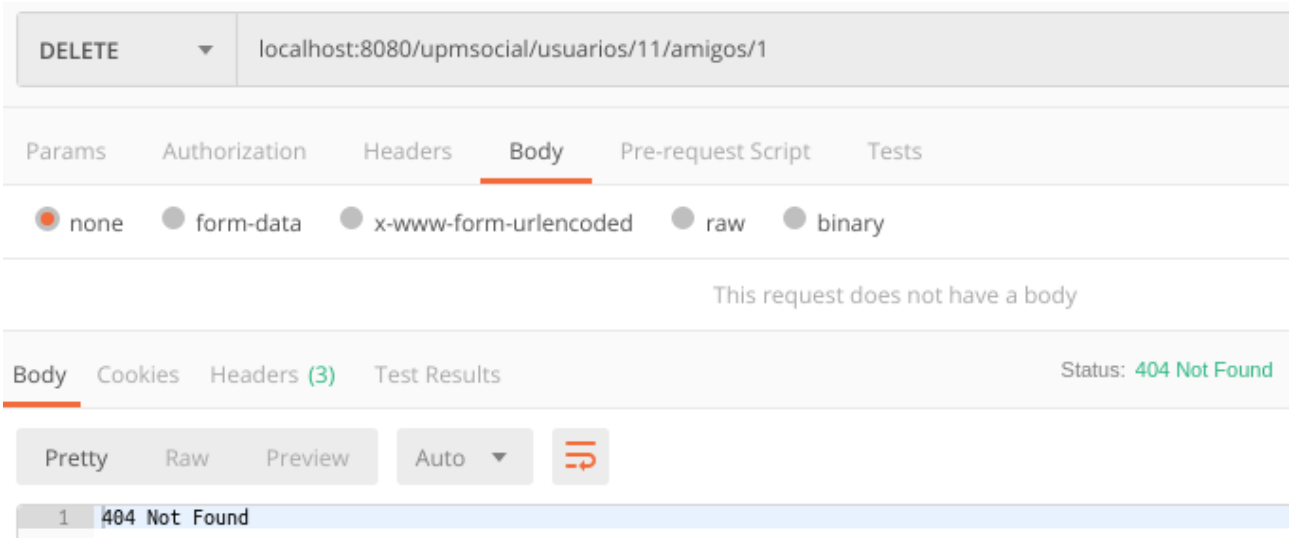
1. DELETE

200 OK



Cuando se borra un amigo satisfactoriamente, se devuelve un *200 OK*.

404 Not Found

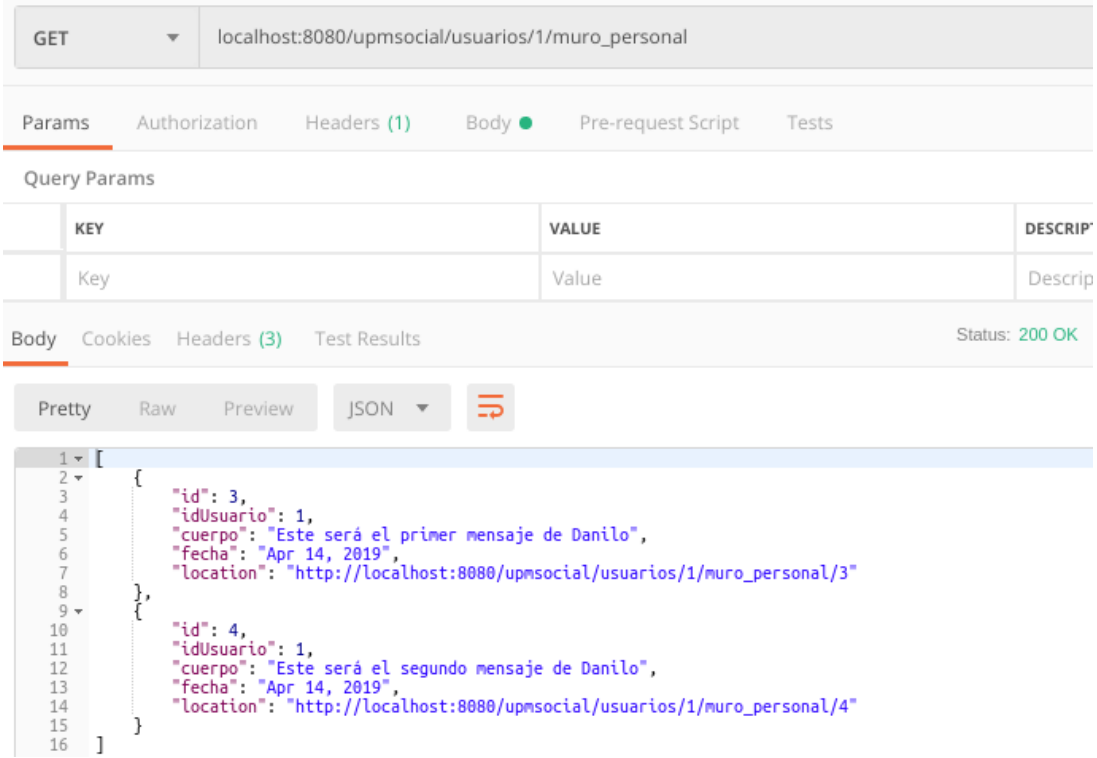


Si la relacion de amistad no existe, o el usuario no existe, se devuelve un codigo *404 Not Found.*

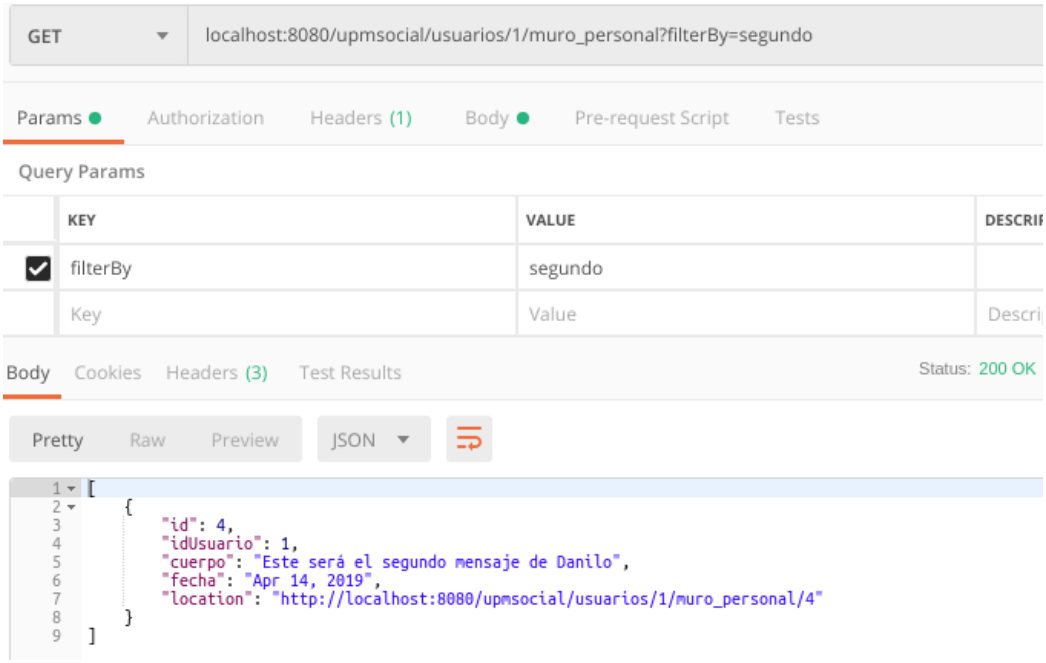
5.5 Muro personal

1. GET

200 OK

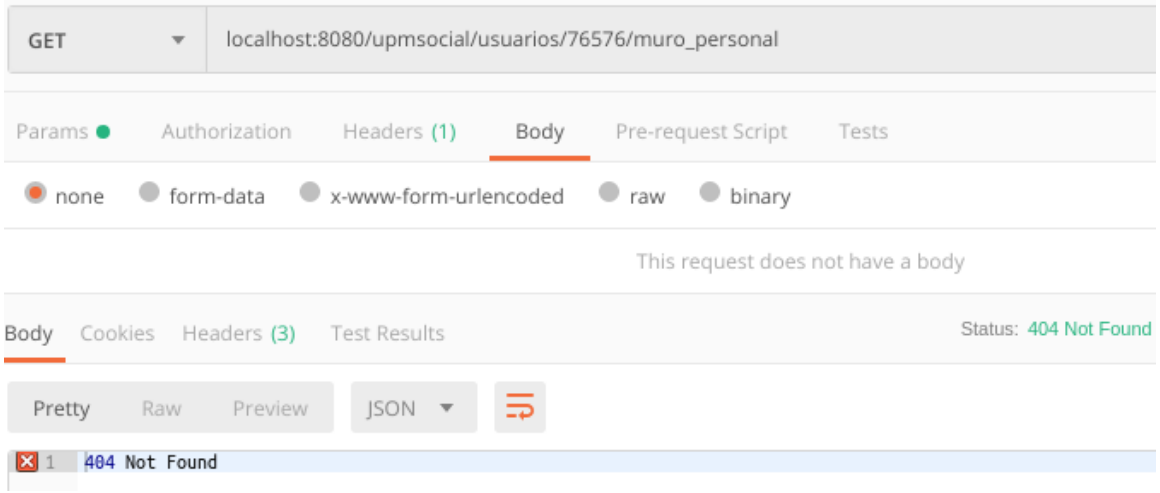


Cuando se listan los mensajes correctamente, se muestra un json de cada uno y un *200 OK*.

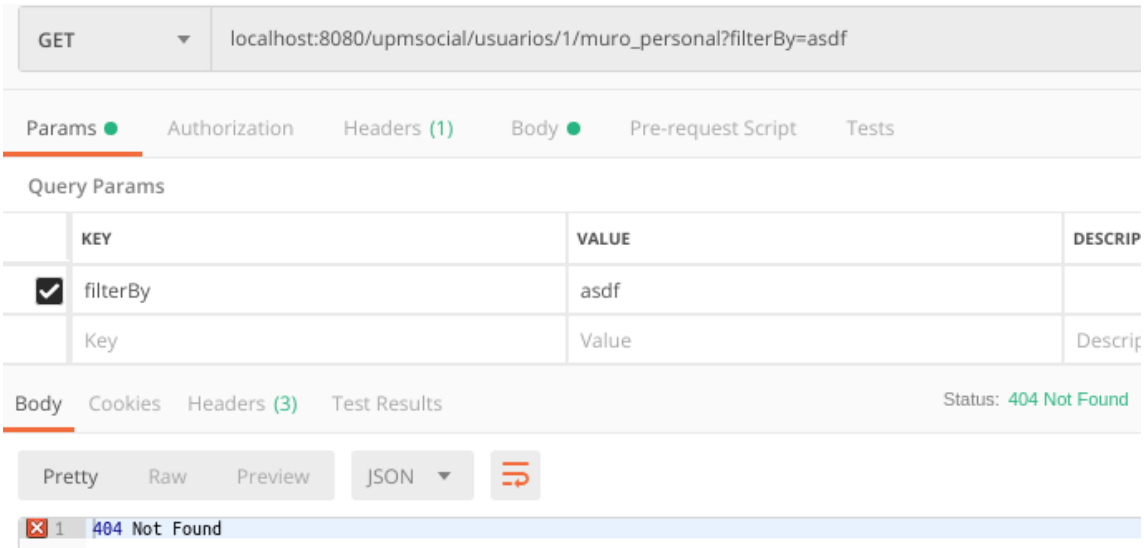


Tambien se puede añadir un filtro para el cuerpo del mensaje.

404 Not Found



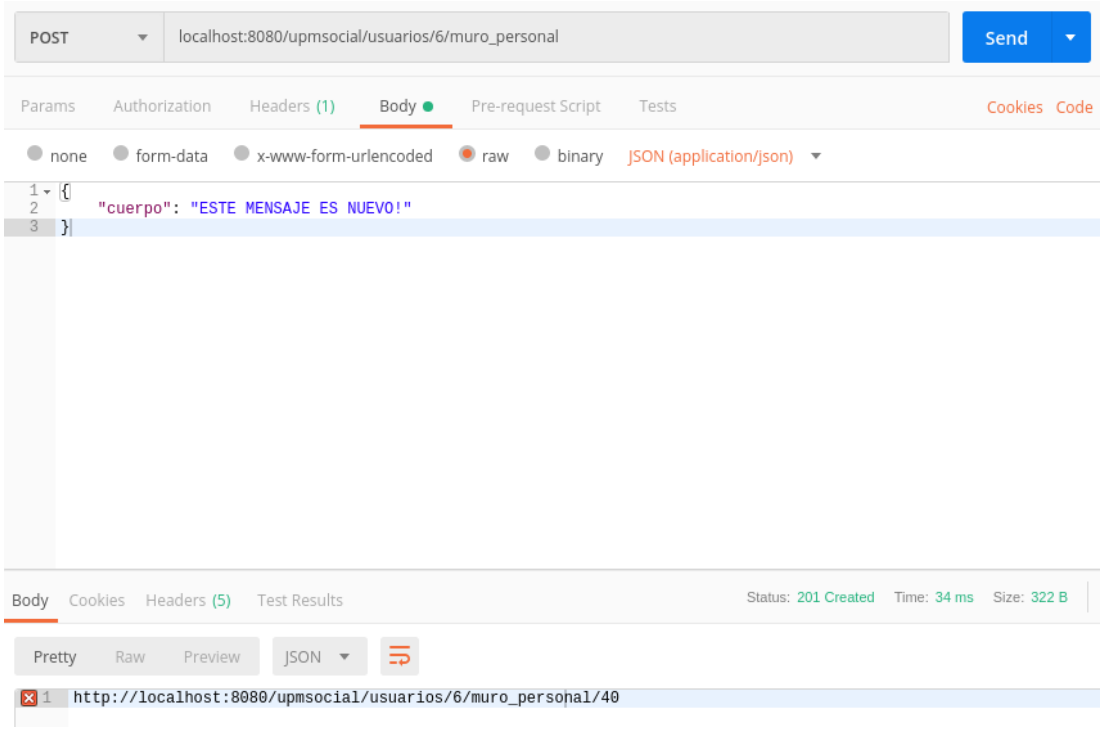
Si el usuario no existe, se devuelve el error *404*.



También si el filtro no encuentra ninguna coincidencia, se obtiene *404 Not Found*.

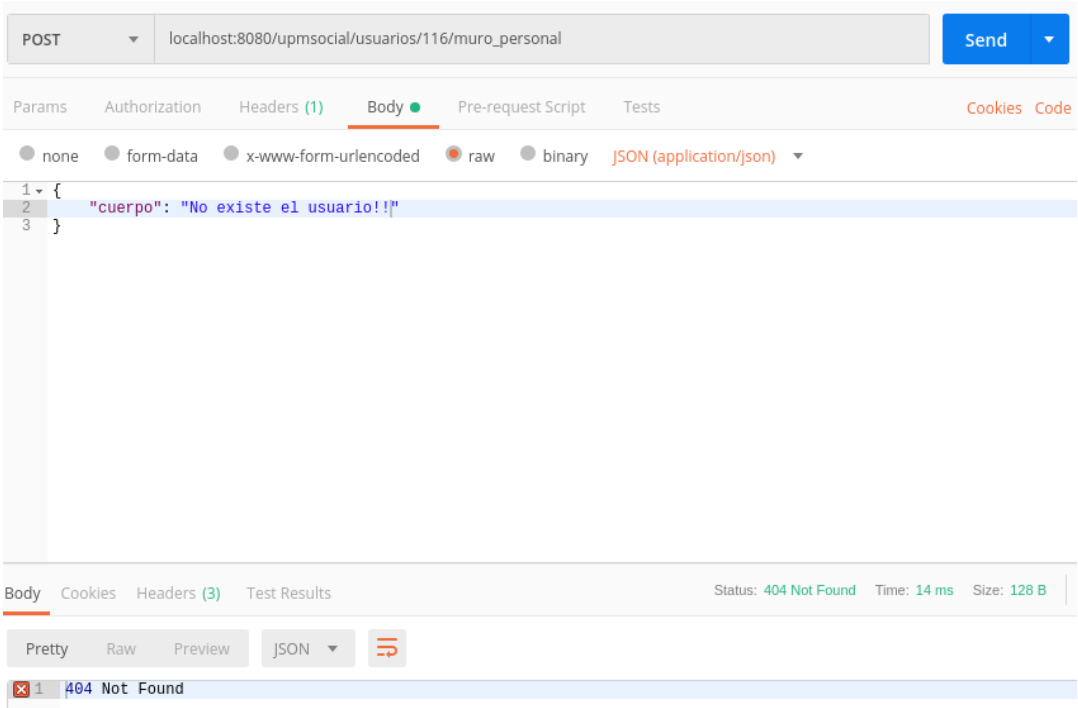
2. POST

201 Created



A la hora de crear un nuevo mensaje en el muro, devolvemos la *URI* del nuevo recurso.

404 Not Found

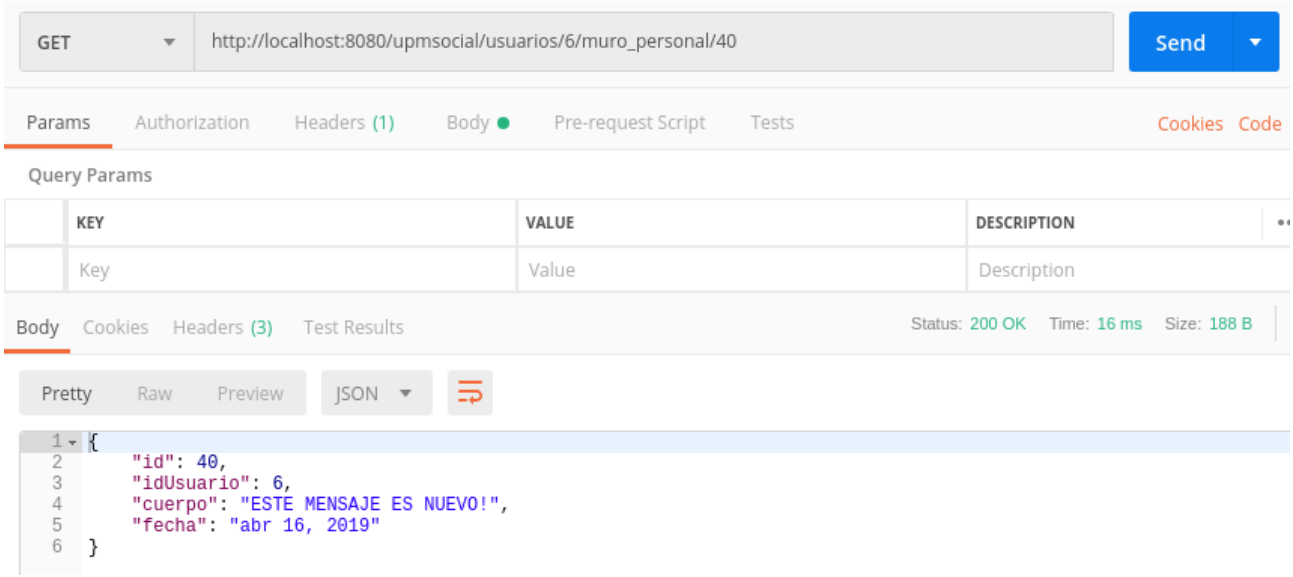


Si el usuario no existe, no se podrá crear un mensaje en el muro.

5.6 Mensaje (concreto) en muro personal

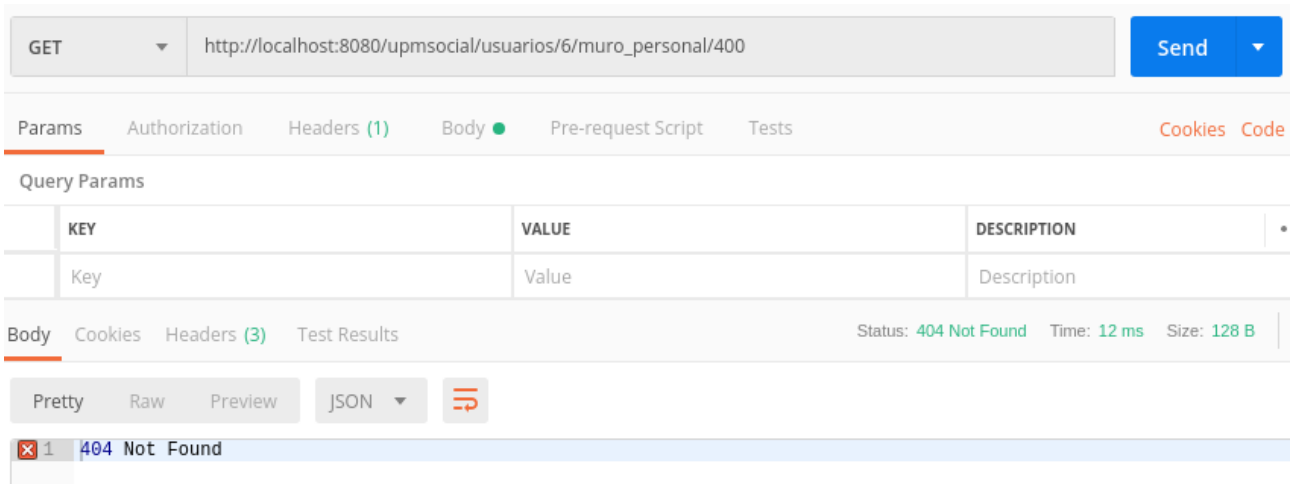
1. GET

200 OK



Como vemos, la *URI* devuelta en el *POST* anterior, es accesible, y muestra los datos del mensaje.

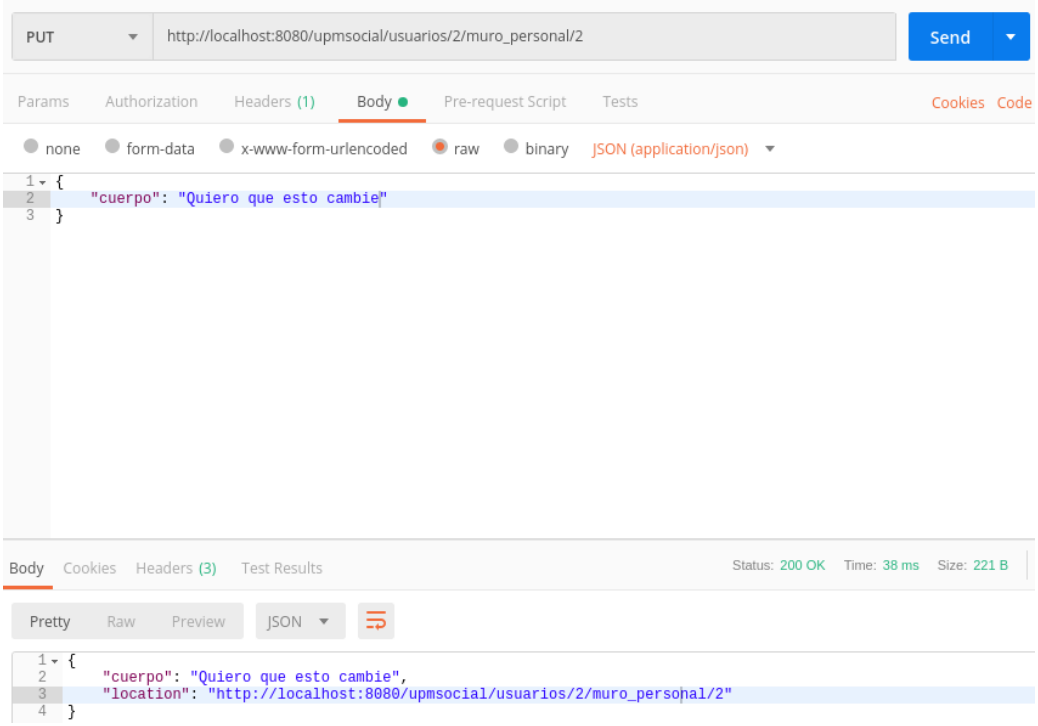
404 Not Found



Si alguno de los dos ID no existe, se devolverá un error de no encontrado.

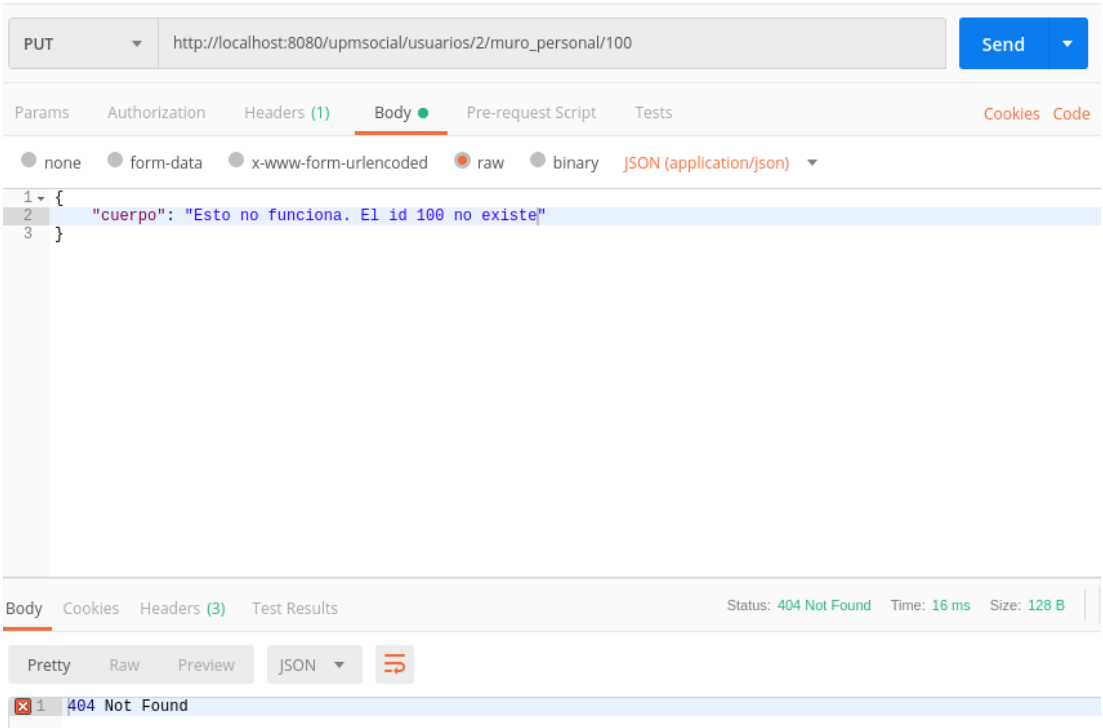
2. PUT

200 OK



Es posible editar un mensaje ya creado mediante un *PUT*. Como respuesta obtendremos el nuevo cuerpo del mensaje y su *URI*.

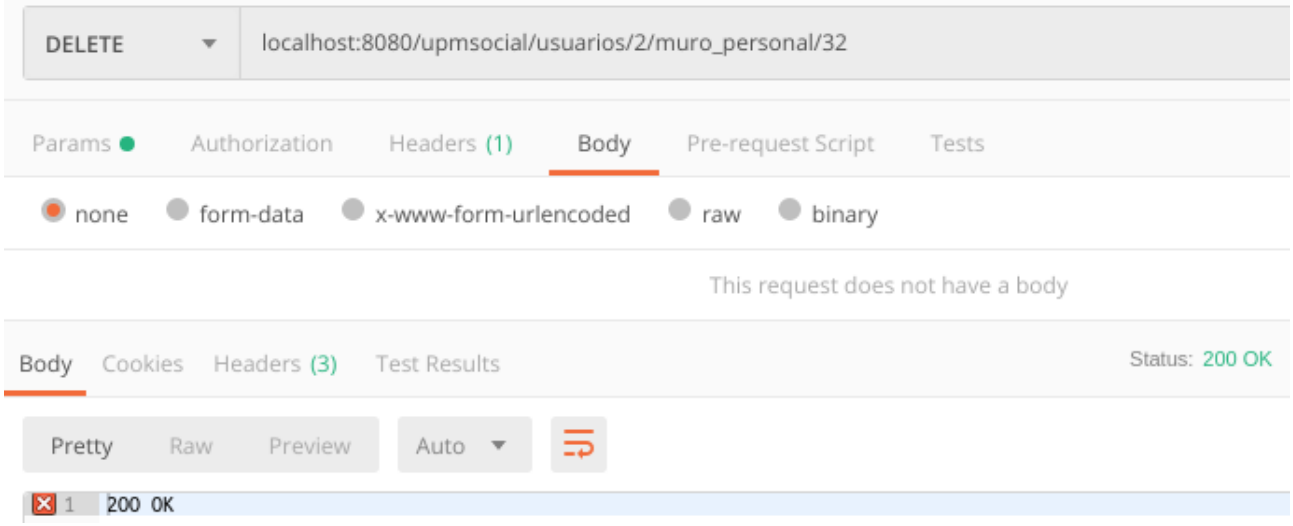
404 Not Found



Si hay algún problema en cuanto a ID, como que el usuario no haya escrito el mensaje, se devolverá un *404 Not Found*.

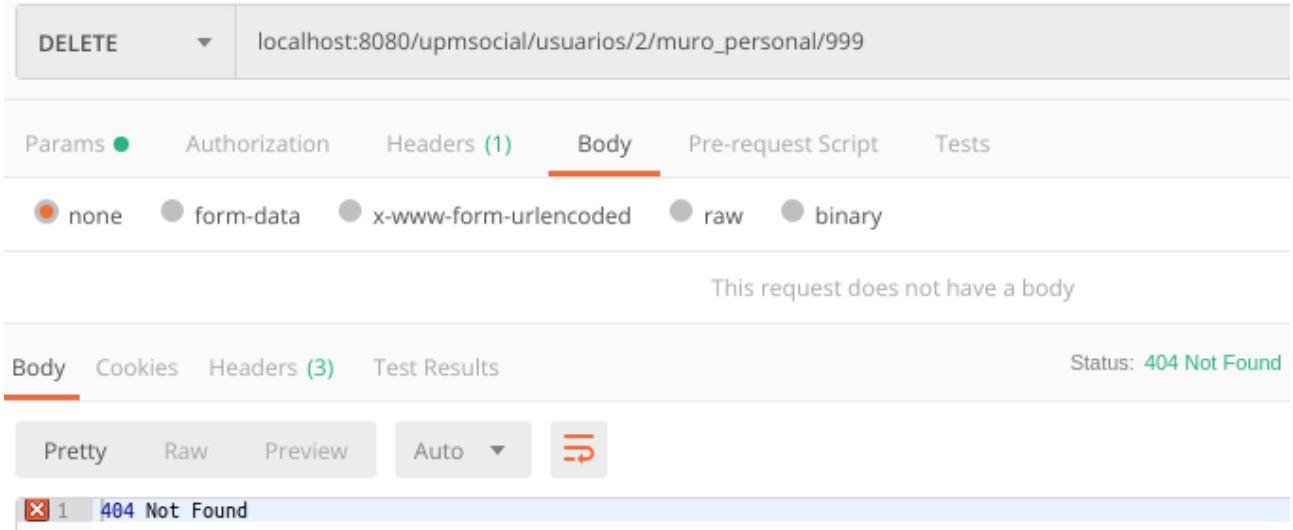
3. DELETE

200 OK



Si el borrado se ejecuta satisfactoriamente, se devuelve un *200 OK*.

404 Not Found

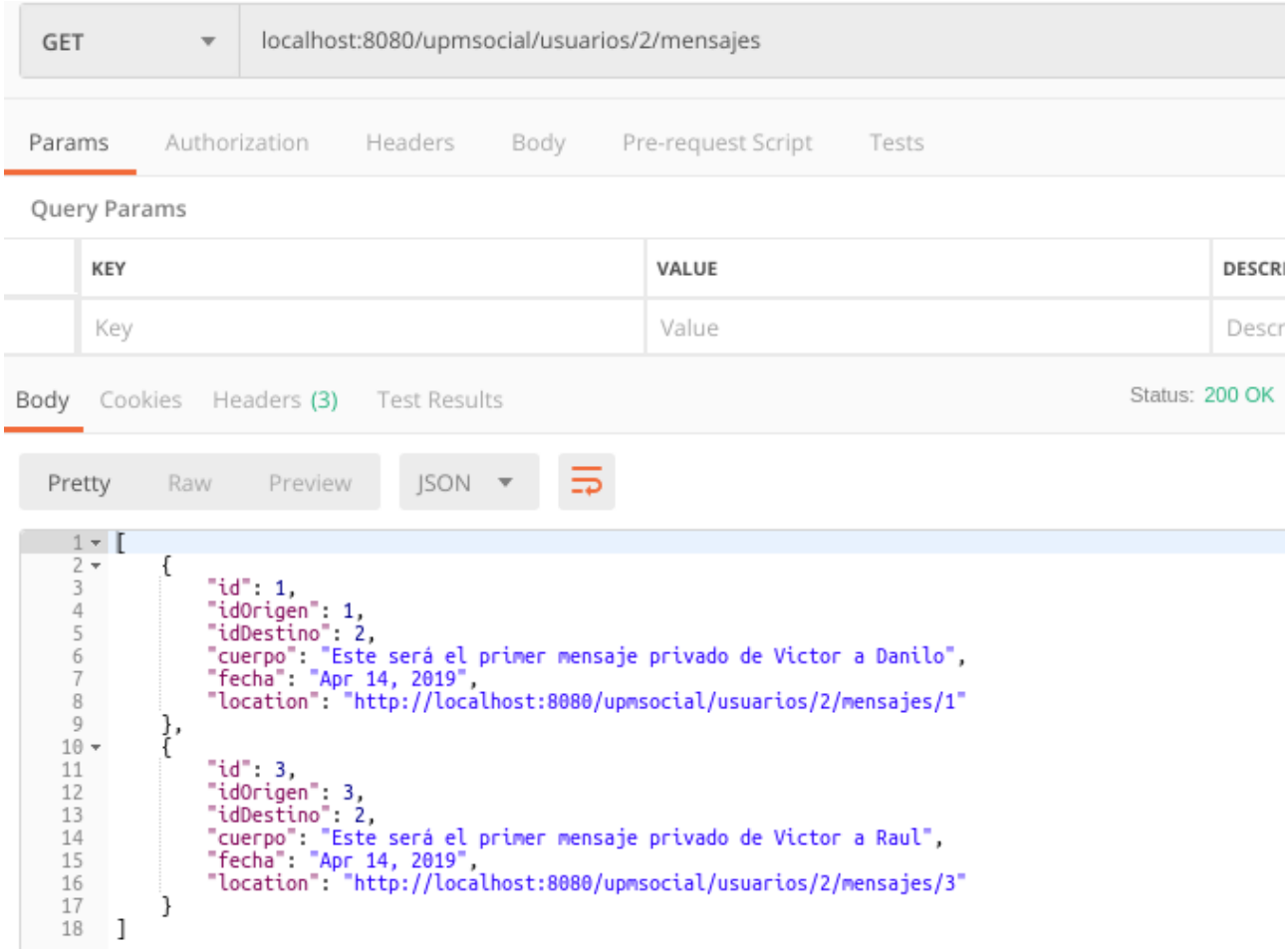


Si el mensaje no existe, se devuelve un *404 Not Found.*

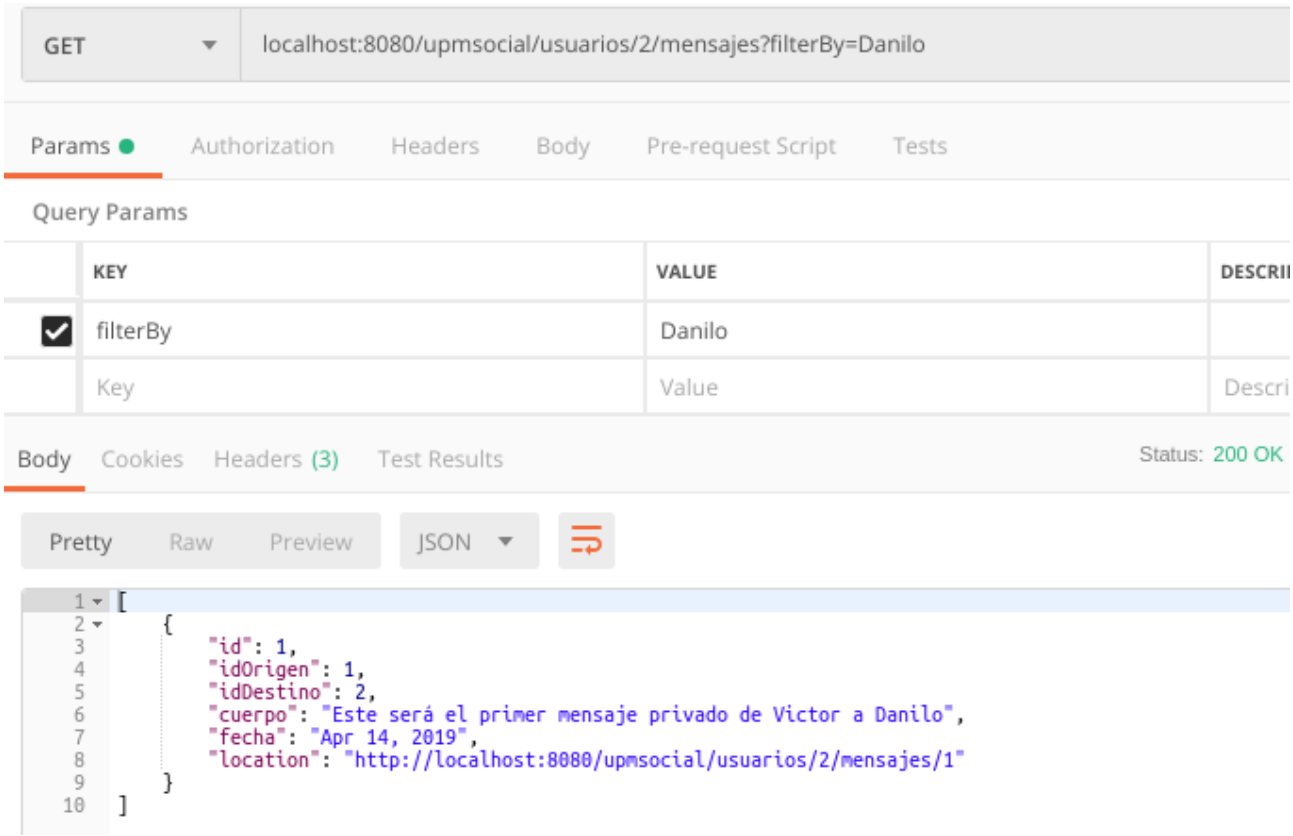
5.7 Mensajes privados

1. GET

200 OK

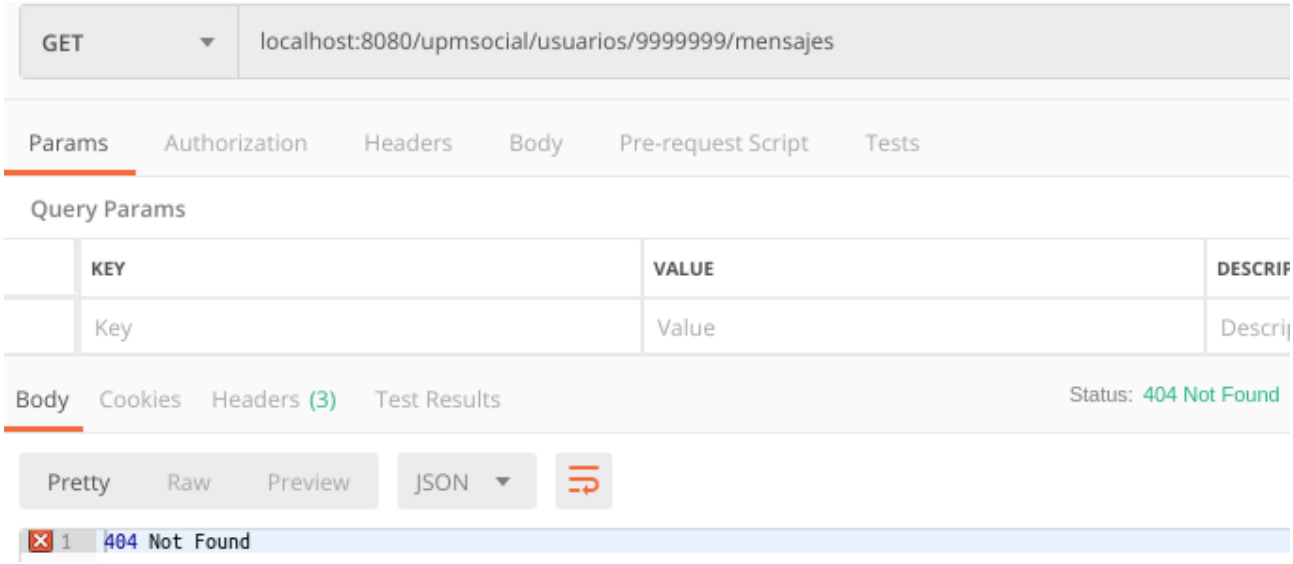


Si se encuentra correctamente la lista de mensajes, se devuelve un *200 OK*.

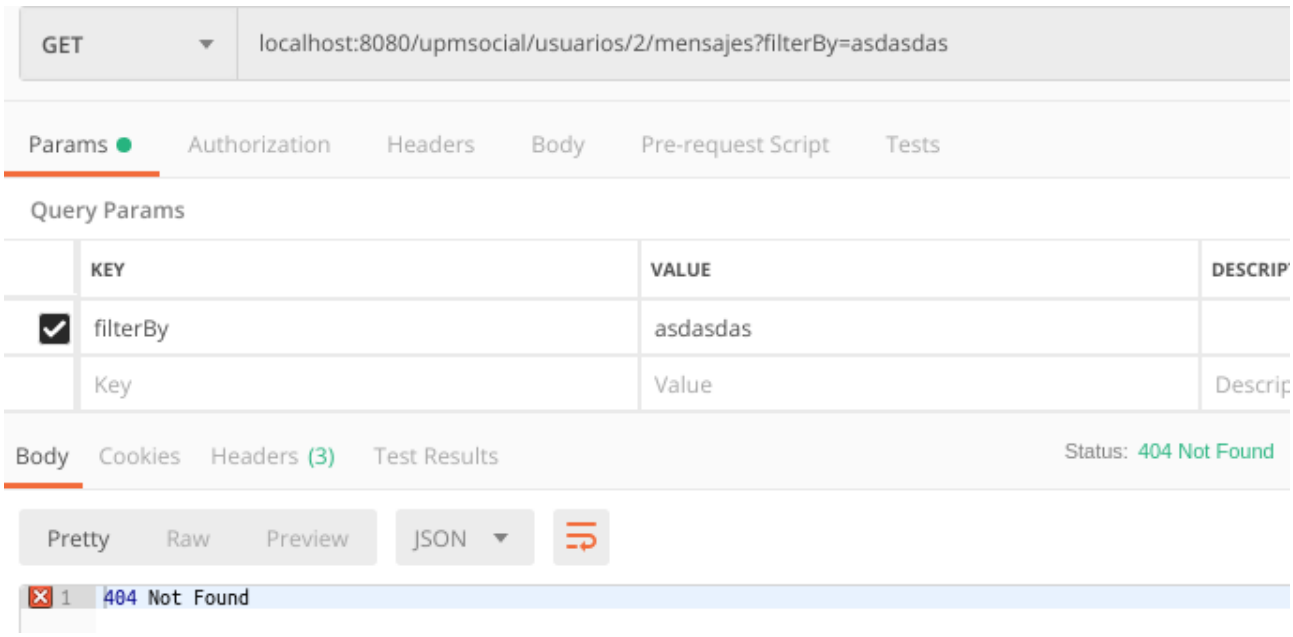


Tambien tenemos un filtro sobre el cuerpo del mensaje.

404 Not Found



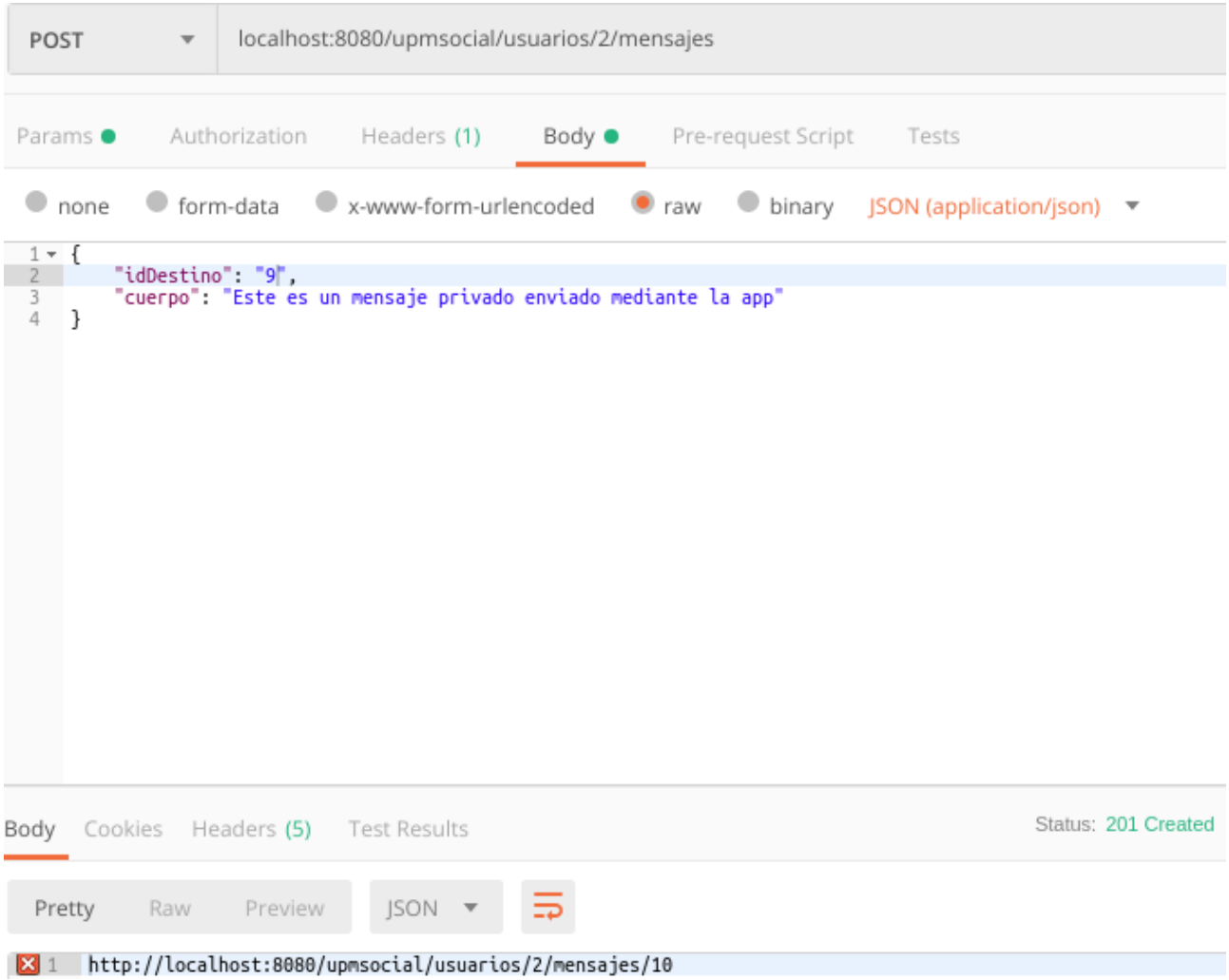
Si el usuario no existe, se devuelve un *404 Not Found*.



Si el filtro no encuentra ninguna coincidencia se devuelve un *404 Not Found*.

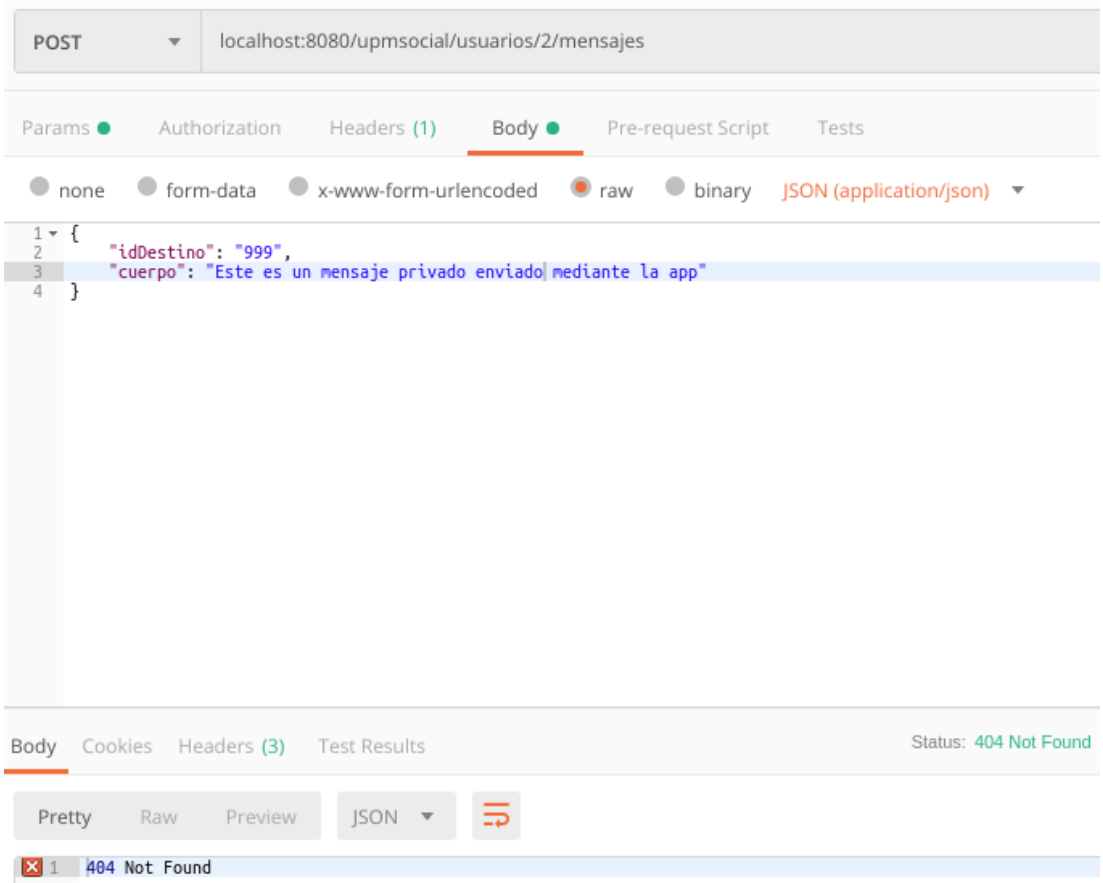
2. POST

201 Created



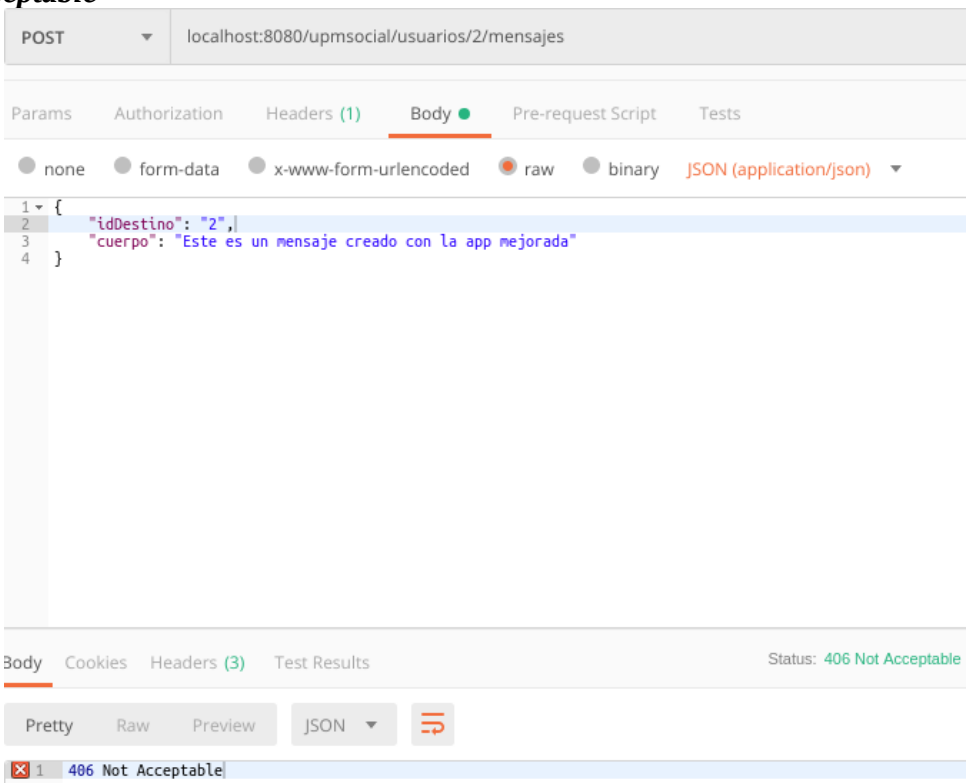
Si el mensaje se envia con éxito, se devuelve un *201 Created*.

404 Not Found



Si alguno de los dos usuarios no existen, se devuelve un *404 Not Found*.

406 Not Acceptable

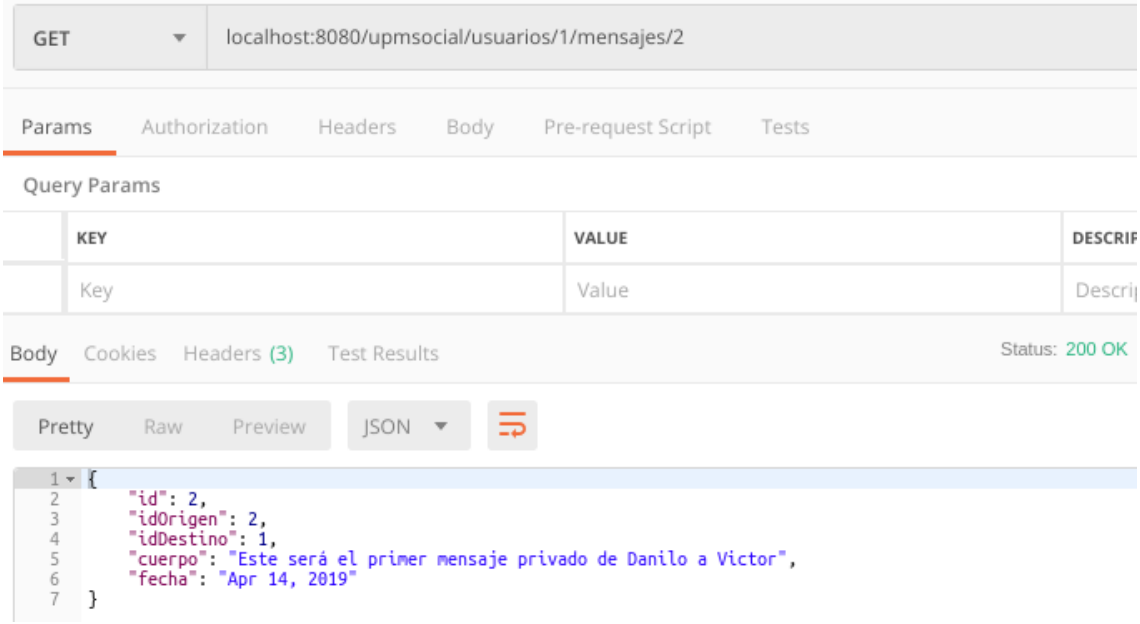


Si el destinatario del mensaje es el mismo usuario que el del origen, se devuelve un *406 Not Acceptable*.

5.8 Mensaje (concreto) en mensajes privados

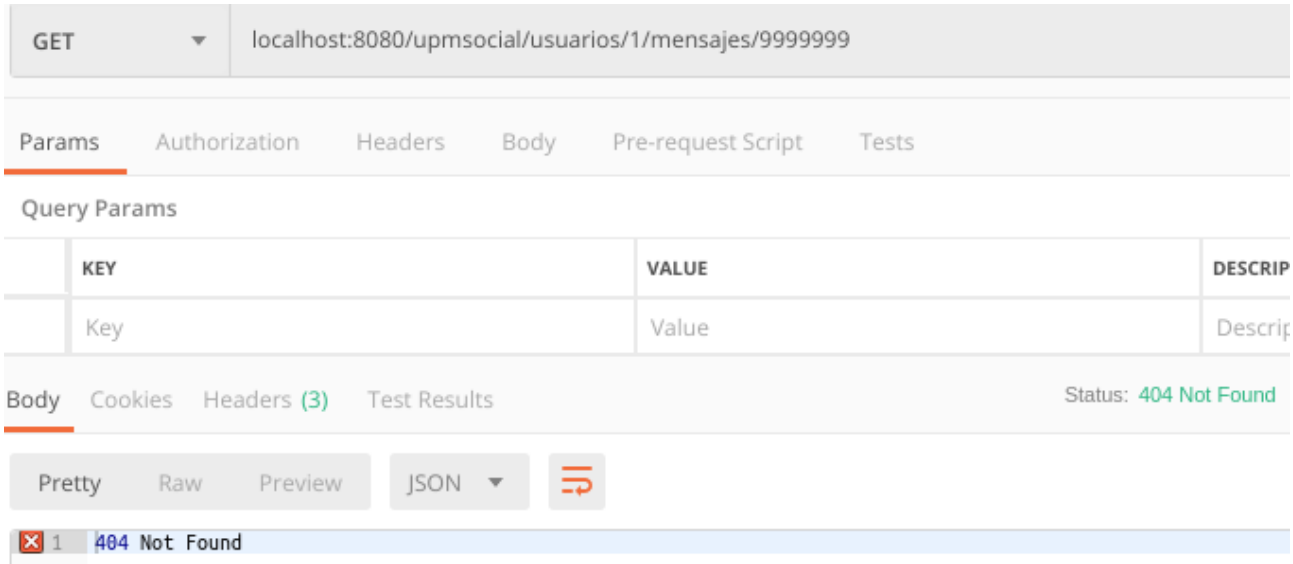
1. GET

200 OK



Si se accede exitosamente a la informacion, se devuelve un *200 OK*.

404 Not Found

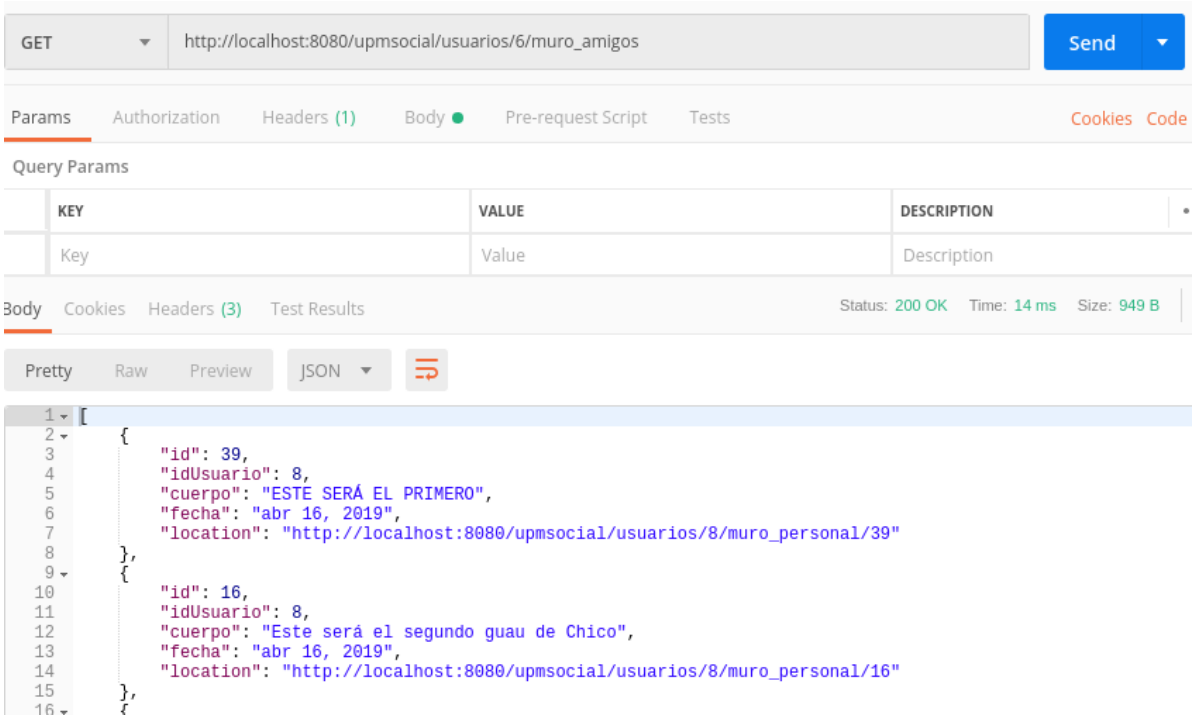


Si el usuario o el mensaje no existen, se devuelve un *404 Not Found*.

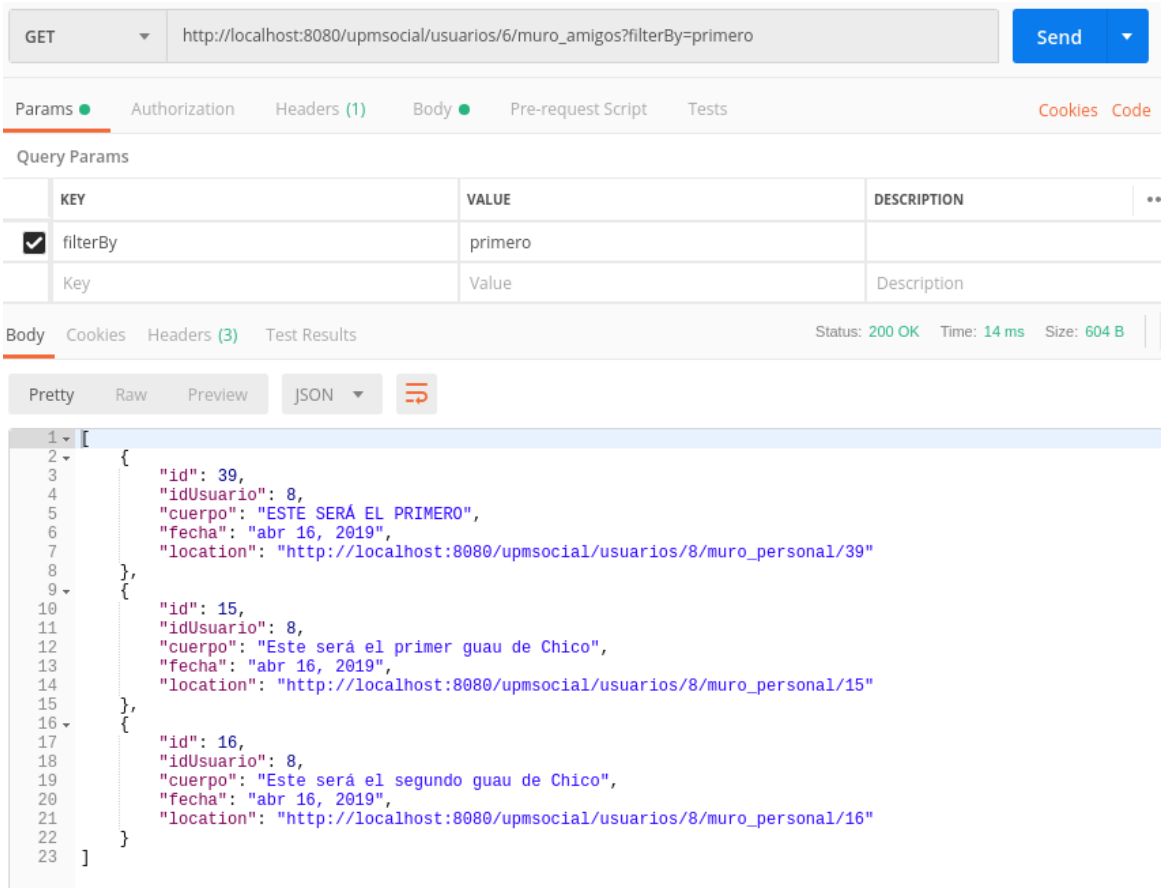
5.9 Muro de amigos

1. GET

200 OK

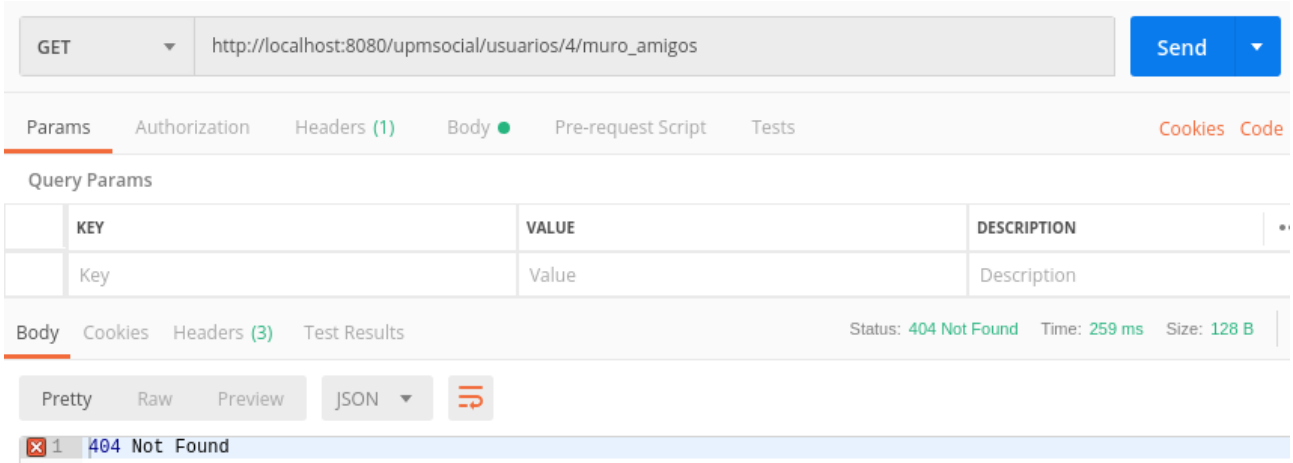


Podemos obtener los mensajes en los muros de nuestros amigos.



Además esa busqueda puede ser filtrada por el cuerpo del mensaje.

404 Not Found

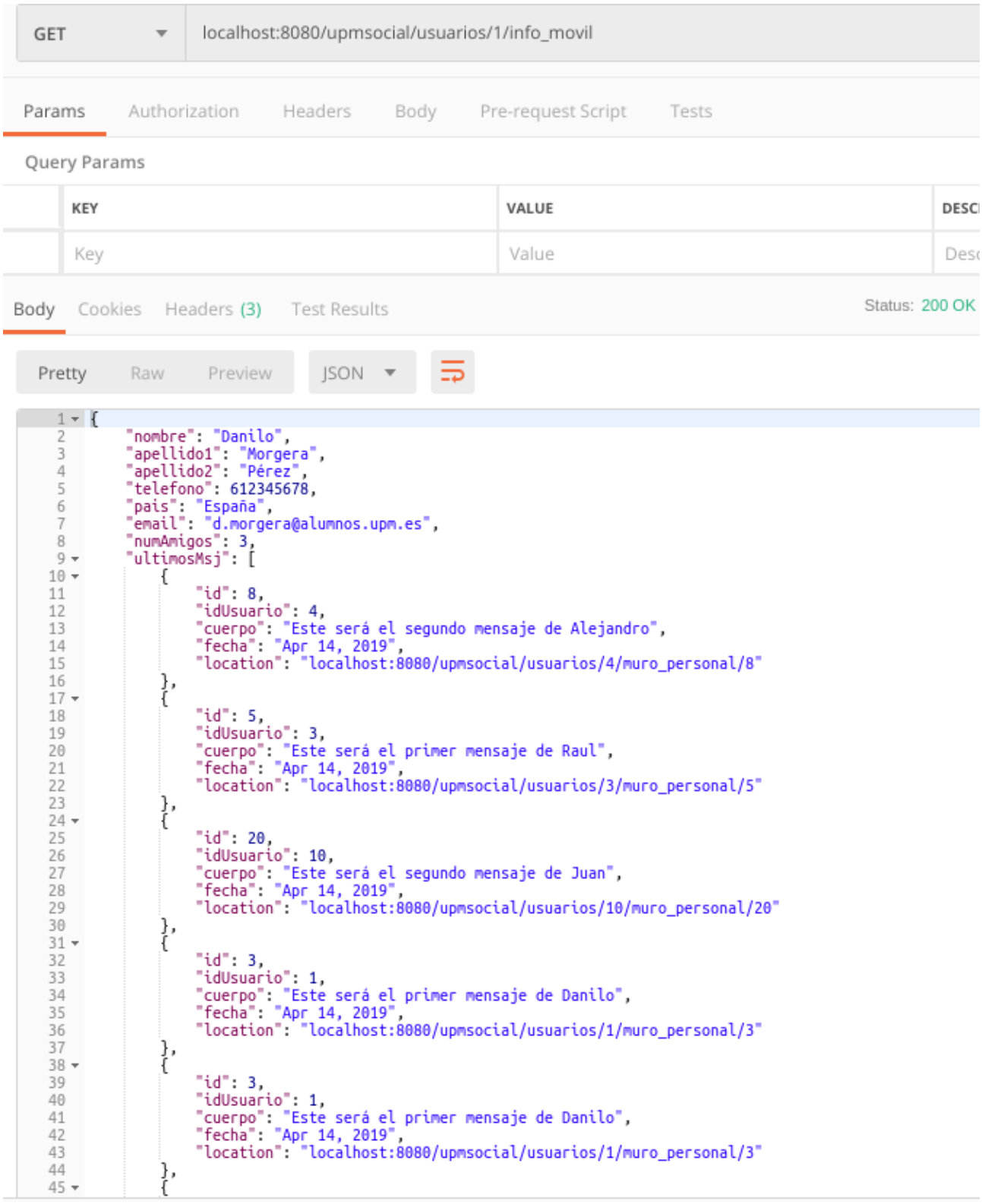


Devolveremos un error *404 Not Found* si el ID no existe, si ese usuario no tiene amigos, o ninguno de sus amigos ha publicado ningún mensaje en el muro.

5.10 Informacion para el movil de un usuario

1. GET

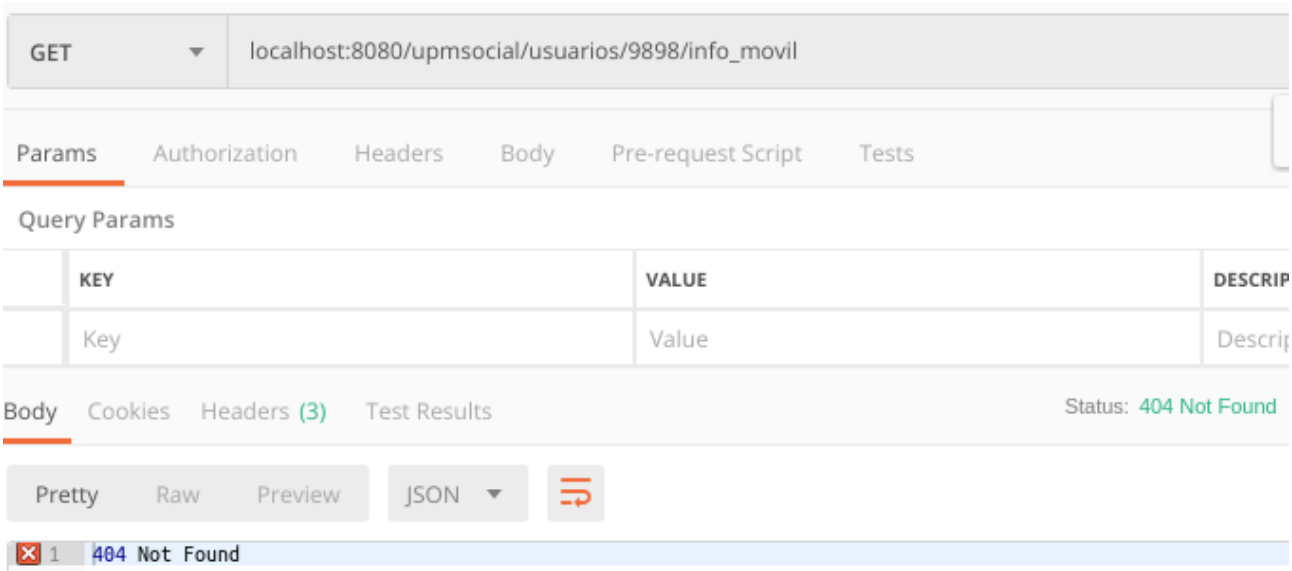
200 OK




```
{
    "nombre": "Danilo",
    "apellido1": "Morgera",
    "apellido2": "Pérez",
    "telefono": 612345678,
    "pais": "España",
    "email": "d.morgera@alumnos.upm.es",
    "numAmigos": 3,
    "ultimosMsj": [
        {
            "id": 8,
            "idUsuario": 4,
            "cuerpo": "Este será el segundo mensaje de Alejandro",
            "fecha": "Apr 14, 2019",
            "location": "localhost:8080/upmsocial/usuarios/4/muro_personal/8"
        },
        {
            "id": 5,
            "idUsuario": 3,
            "cuerpo": "Este será el primer mensaje de Raul",
            "fecha": "Apr 14, 2019",
            "location": "localhost:8080/upmsocial/usuarios/3/muro_personal/5"
        },
        {
            "id": 20,
            "idUsuario": 10,
            "cuerpo": "Este será el segundo mensaje de Juan",
            "fecha": "Apr 14, 2019",
            "location": "localhost:8080/upmsocial/usuarios/10/muro_personal/20"
        },
        {
            "id": 3,
            "idUsuario": 1,
            "cuerpo": "Este será el primer mensaje de Danilo",
            "fecha": "Apr 14, 2019",
            "location": "localhost:8080/upmsocial/usuarios/1/muro_personal/3"
        },
        {
            "id": 3,
            "idUsuario": 1,
            "cuerpo": "Este será el primer mensaje de Danilo",
            "fecha": "Apr 14, 2019",
            "location": "localhost:8080/upmsocial/usuarios/1/muro_personal/3"
        },
        {
```

Si el usuario existe, se devolvera un JSON con toda la informacion relevante, asi como un *200 OK*.

404 Not Found

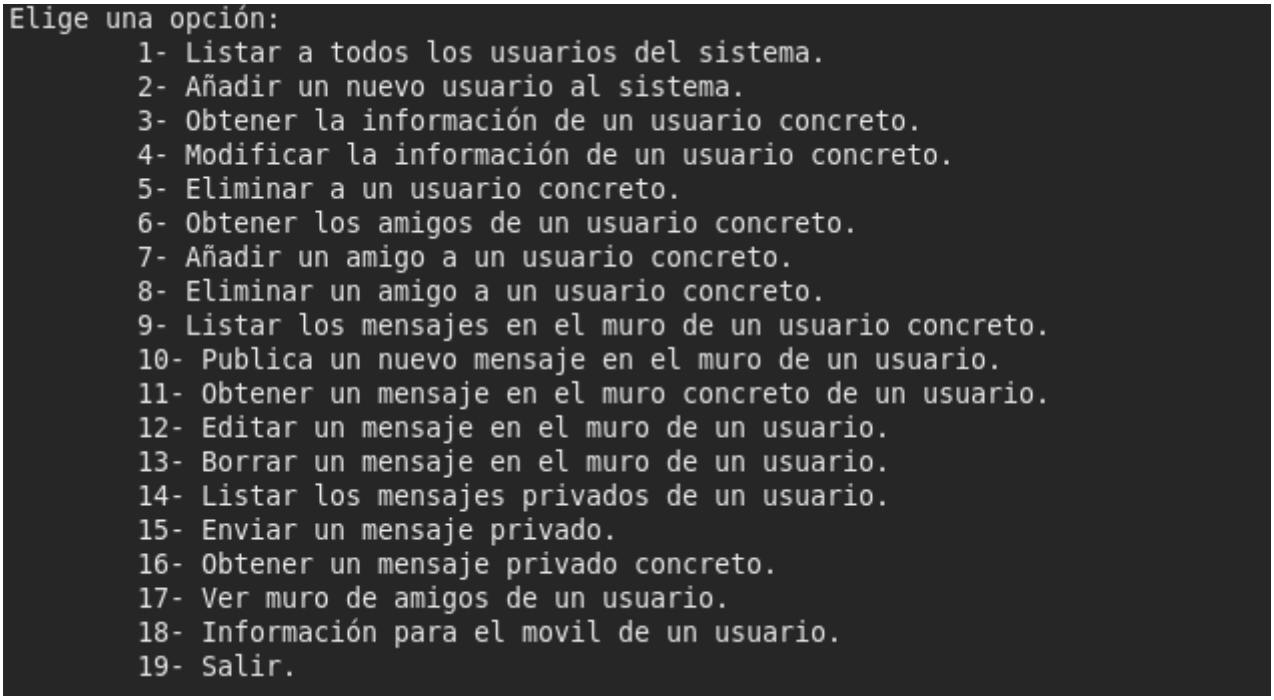


Si el usuario no existe, se devuelve un *404 Not Found*.

6 Capturas de la ejecución del cliente

Para el cliente, se ha diseñado un cliente totalmente funcional a través del terminal.

Donde se tienen todas las opciones de operaciones que nuestra *API REST* puede realizar.



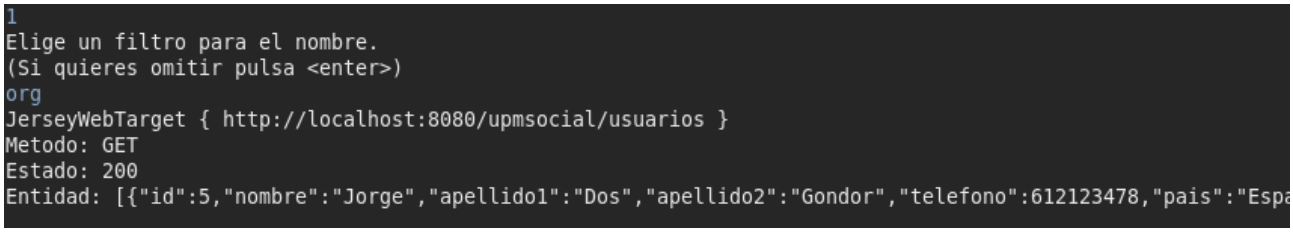
A continuación se mostrará la traza que imprime nuestro cliente a la hora de sellecionar las distintas opciones.

No se mostrarán todas las salidas posibles (distintos códigos de cada función), ya que en el apartado anterior se han mostrado todas las posibles salidas de cada recurso. Se mostrarán varios ejemplos, mostrando diversos códigos de funcionamiento.

Cada operación devuelve la *URI* asociada, el método que se usa, el código de operación y lo que devuelve tal método.

Nota: El primer dígito que se muestra en cada imágen corresponde con la opreación selecionada de la pantalla de inicio.

6.1 Listar a todos los usuarios del sistema.



Devuelve en un *JSON* una lista con todos los usuarios del sistema. En esete caso, la lista está filtrada por “org”.

6.2 Añadir un nuevo usuario al sistema.

```
2
Escribe un nombre.
jesus
Escribe un apellido.
diez
Escribe otro apellido.
ventura
Escribe un email.
jdv@gmail.es
Escribe un pais.
italia
Escribe un numero de telefono.
435365677
JerseyWebTarget { http://localhost:8080/upmsocial/usuarios }
Metodo: POST
Estado: 201
Entidad: http://localhost:8080/upmsocial/usuarios/16
```

Devuelve la *URI* del nuevo recurso

6.3 Obtener la información de un usuario concreto.

```
3
Escribe un id.
300
JerseyWebTarget { http://localhost:8080/upmsocial/usuarios/300 }
Metodo: GET
Estado: 404
Entidad: 404 Not Found
```

En nuestro sistema no existe ningún usuario con id 300.

6.4 Modificar la información de un usuario concreto.

```
4
Escribe un id.
3
Escribe un nombre.

Escribe un apellido.
ape
Escribe otro apellido.
llido
Escribe un email.
invalido
Escribe un pais.

Escribe un numero de telefono.
12345675
JerseyWebTarget { http://localhost:8080/upmsocial/usuarios/3 }
Metodo: PUT
Estado: 406
Entidad: 406 Not Acceptable
```

No se se acepta esta petición, ya que hay datos vacíos (inválidos).

6.5 Eliminar a un usuario concreto.

```
5
Escribe un id.
3
JerseyWebTarget { http://localhost:8080/upmsocial/usuarios/3 }
Metodo: DELETE
Estado: 200
Entidad: 200 OK
```

Un usuario existente eliminado correctamente devuelve un *200 OK*.

6.6 Obtener los amigos de un usuario concreto.

```
6
Escribe un id.
2
Elige un filtro para el nombre.
(Si quieres omitir pulsa <enter>)

Elige inicio.
(Si quieres omitir pulsa <enter>)

Elige final.
(Si quieres omitir pulsa <enter>)

JerseyWebTarget { http://localhost:8080/upmsocial/usuarios/2/amigos }
Metodo: GET
Estado: 200
Entidad: [{"id":1,"nombre":"Danilo","apellido1":"Morgera","apellido2":"Pérez","telefono":612345678,"pais":"España","email":"d.morgera
```

A la hora de solicitar los amigos de un usuario, se devuelve un *JSON* con todos los amigos de un usuario. En este caso, se han omitido todos los filtros.

6.7 Añadir un amigo a un usuario concreto.

```
7
Escribe el id del usuario.
11
Escribe el id del amigo.
7
JerseyWebTarget { http://localhost:8080/upmsocial/usuarios/11/amigos }
Metodo: POST
Estado: 201
Entidad: http://localhost:8080/upmsocial/usuarios/7
```

Al crear un nuevo amigo, se devuelve la *URI* del nuevo amigo (se devuelve la *URI* del usuario que es amigo nuevo).

6.8 Eliminar un amigo a un usuario concreto.

```
8
Escribe el id del usuario.
11
Escribe el id del amigo.
1
JerseyWebTarget { http://localhost:8080/upmsocial/usuarios/11/amigos/1 }
Metodo: DELETE
Estado: 404
Entidad: 404 Not Found
```

Cuando no existe una amistad, no se puede eliminar, y devuelve un error *404.*

6.9 Listar los mensajes en el muro de un usuario concreto.

```
9
Escribe un id.
2
Elige un filtro para el cuerpo del mensaje.
(Si quieres omitir pulsa <enter>)
Segundo
JerseyWebTarget { http://localhost:8080/upmsocial/usuarios/2/muro_personal }
Metodo: GET
Estado: 200
Entidad: [{"id":2,"idUsuario":2,"cuerpo":"Este será el segundo mensaje de Victor","fecha":"abr 16, 2019", "location": "http://localh
```

Al obtener los mensajes en el muro de un usuario, se nos devuelve en formato *JSON* el mensaje en sí y su correspondiente *URI.* Se ha filtrado por la palabra “Segundo” en el cuerpo del mensaje.

6.10 Publicar un nuevo mensaje en el muro de un usuario

```
10
Escribe un id.
2
Escribe el texto del mensaje.
Este mensaje viene desde el cliente
JerseyWebTarget { http://localhost:8080/upmsocial/usuarios/2/muro_personal }
Metodo: POST
Estado: 201
Entidad: http://localhost:8080/upmsocial/usuarios/2/muro_personal/53
```

A la hora de publicar un nuevo mensaje, si el usuario existe, se devolverá la *URI* del nuevo recurso (mensaje).

6.11 Obtener un mensaje del muro concreto

```
11
Escribe el id del usuario.
2
Escribe el id del mensaje.
55
JerseyWebTarget { http://localhost:8080/upmsocial/usuarios/2/muro_personal/55 }
Metodo: GET
Estado: 404
Entidad: 404 Not Found
```

Para obtener un mensaje concreto, se comprueba que los datos sean correctos, es decir, que el usuario haya escrito ese mensaje. Si no se cumple, se devuelve un *404 Not Found*.

6.12 Editar un mensaje del muro de un usuario

```
12
Escribe el id del usuario.
2
Escribe el id del mensaje.
53
Escribe el texto del mensaje.
Soy el de antes. Quiero editar!
JerseyWebTarget { http://localhost:8080/upmsocial/usuarios/2/muro_personal/53 }
Metodo: PUT
Estado: 200
Entidad: {"cuerpo":  "Soy el de antes. Quiero editar!" , "location": "http://localhost:8080/upmsocial/usuarios/2/muro_personal/53"}
```

A la hora de editar, si ha habido éxito, se devuelve un *JSON* con el nuevo cuerpo del mensaje y la *URI* del recurso.

6.13 Borra un mensaje del muro concreto

```
13
Escribe el id del usuario.
2
Escribe el id del mensaje.
1
JerseyWebTarget { http://localhost:8080/upmsocial/usuarios/2/muro_personal/1 }
Metodo: DELETE
Estado: 200
Entidad: 200 OK
```

Si los ID concuerdan, se hace un *DELETE* y se devuelve un *200 OK*.

6.14 Listar los mensajes privados de un cliente

```
14
Escribe un id.
2
Elige un filtro para el cuerpo del mensaje.
(Si quieres omitir pulsa <enter>)
primer
JerseyWebTarget { http://localhost:8080/upmsocial/usuarios/2/mensajes }
Metodo: GET
Estado: 200
Entidad: [{"id":1,"idOrigen":1,"idDestino":2,"cuerpo":"Este será el primer mensaje privado de Victor
```

Se ha implementado el obtener una lista de mensajes privados con un filtro opcional. En este caso se ha buscado todos los mensajes que contengan la palabra “primer”.

6.15 Enviar un mensaje privado

```
15
Escribe un id de origen.
1
Escribe un id de destino.
2
Escribe el texto del mensaje.
hola amigo
JerseyWebTarget { http://localhost:8080/upmsocial/usuarios/1/mensajes }
Metodo: POST
Estado: 201
Entidad: http://localhost:8080/upmsocial/usuarios/1/mensajes/10
```

Si ambos ID existen en la base de datos, se podrá enviar un nuevo mensaje privado.

6.16 Obtener un mensaje del muro de un usuario

```
16
Escribe el id del usuario.
2
Escribe el id del mensaje.
1
JerseyWebTarget { http://localhost:8080/upmsocial/usuarios/2/mensajes/1 }
Metodo: GET
Estado: 200
Entidad: {"id":1,"idOrigen":1,"idDestino":2,"cuerpo":"Este será el primer mensaje privado de Victor a Dani
```

Se obtiene el mensaje, siempre y cuando los ID concuerden.

6.17 Ver el muro de los amigos de un usuario

```
17
Escribe un id.
2
Elige un filtro para el cuerpo del mensaje.
(Si quieres omitir pulsa <enter>)
xxdedxxx
JerseyWebTarget { http://localhost:8080/upmsocial/usuarios/2/muro_amigos }
Metodo: GET
Estado: 404
Entidad: 404 Not Found
```

Si el usuario no existe, no tiene amigos, o el filtro no encuentra ningún mensaje, se devuelve un *404 Not Found.*

6.18 Información para el movil de un usuario

```
18
Escribe un id.
3
JerseyWebTarget { http://localhost:8080/upmsocial/usuarios/3/info_movil }
Metodo: GET
Estado: 200
Entidad: {"nombre":"Raul","apellido1":"Prieto","apellido2":"Acedo","telefono":611232678,"pais":"España","emai
```

Por ultimo, a la hora de mostrar toda la información para un dispositivo móvil, enviamos un *JSON* con todos los datos.

7 Referencias

[1]: *Transferencia de Estado Representacional* | Wikipedia Online, disponible en: https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional

[2]: *¿Qué es la arquitectura REST?* | GaussWebApp Online, disponible en:

https://gausswebapp.com/arquitectura-rest.html

[3]: *Características REST* | BBVAopen4u Online, disponible en:

https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos

[4]: *Ventajas que ofrece REST para el desarrollo* | BBVAopen4u Online, disponible en:

https://bbvaopen4u.com/es/actualidad/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos