

Open Opened 3 days ago by  **Rubén Montero**

## Subir una idea

### Resumen

- Explicaremos cómo funciona la petición **POST** con la que los usuarios suben ideas
- Veremos por qué no se puede falsificar. Sólo cada usuario sube sus ideas
- La implementaremos

### Descripción

El **corazón** () de nuestro **IdeAPI** es la petición **POST** que permite a los usuarios *subir* (crear) una nueva idea en la base de datos.

En nuestros proyectos anteriores, los **POST** se consideraban anónimos. No requeríamos autenticación.

Pero ahora...

### Autenticación

¡Es muy sencillo!

Lo que podría ser un **POST** para subir una idea, sin información de *autor*:

```
POST /v1/categories/2/ideas

{
  "new_idea_title": "Bajar el precio de la máquina de café",
  "content": "¡Es muy caro!"
}
```

Llevará en las **cabeceras HTTP** un *token de sesión*.

```
POST /v1/categories/2/ideas
Api-Session-Token: 51a3bc2ada4909f5dff50e468aa887ca2a0a62b

{
  "new_idea_title": "Bajar el precio de la máquina de café",
  "content": "¡Está muy caro!"
}
```

Nótese que hay una serie de **cabeceras estándar**. Pero está permitido usar *cabeceras propias*, como nuestra **Api-Session-Token**.

### ¿Y cómo se envían desde un cliente HTTP las **cabeceras**?

Eso es preocupación de quien esté implementando el cliente HTTP (e.g: App Android, JavaScript de una web,...)

Nostros, para nuestras pruebas, podemos usar **cURL** con **--header**:

```
curl -X POST http://localhost:8000/v1/categories/2/ideas --header "Api-Session-Token: 51a3bc2ada4909f5dff50e468aa887ca2a0a62b"
```

### ¿Y cómo se *interpretan* las cabeceras HTTP desde el servidor?

Con Django, basta con usar **.headers**:

```
def my_endpoint(request):
    print(request.headers['Api-Session-Token'])

    # Como headers es un diccionario, podemos
    # acceder a sus clave-valor también usando
    # .get y un valor por defecto
    session_header = request.headers.get('Api-Session-Token', None)
    if session_header is not None:
        print("Tenemos un token del cliente HTTP. Es: " + session_header)
```

```

else:
    print("¡Este cliente no está autenticado!")

```

## ¡Autenticación entendida!

Genial.

Ahora, definamos la **petición** que vamos a implementar:

**/v1/categories/{categoryId}/ideas**

**POST**

**Resumen:**

Crea una nueva idea

**Parámetros**

Nombre	En	Descripción	¿Obligatorio?	Tipo
Api-Session-Token	headers	Token de sesión que identifica y autentica a un usuario	Sí	string
{categoryId}	path	ID de la categoría a la que pertenecerá la idea	Sí	int
new_idea_title	body (json)	Breve descripción de la nueva idea	Sí	string
content	body (json)	Descripción extendida de la idea	Sí	string

**Respuestas**

Código	Descripción
201	La idea se ha creado con éxito
400	La petición HTTP no se envió con todos los parámetros en el JSON
401	El <i>token</i> de sesión no se ha enviado o no es válido
404	La categoría identificada por <code>{categoryId}</code> no existe

---

## ¡Pues vamos allá!

**endpoints.py**

**Añadamos** una nueva función.

Organizaremos el control del método HTTP de la siguiente manera, ya que en el futuro implementaremos un **GET**:

```

@csrf_exempt
def ideas(request, category_id):
    if request.method == 'POST':
        # El usuario va a añadir una idea a La base de datos
    elif request.method == 'GET':
        # El usuario quiere consultar las ideas de la categoría con id == category_id
        pass
    else:
        return JsonResponse({'error': 'HTTP method unsupported'}, status=405)

```

**Si las ideas fueran anónimas...**

...bastaría con:

- Recuperar la categoría de la base de datos:

```

# El usuario va a añadir una idea a La base de datos

```

```

try:
    category = Category.objects.get(id=category_id)
except Category.DoesNotExist:
    return JsonResponse({"error": "Category was not found"}, status=404)

```

- Procesar la información del cuerpo JSON de la petición

```

body_json = json.loads(request.body)
try:
    json_title = body_json['new_idea_title']
    json_description = body_json['content']
except KeyError:
    return JsonResponse({"error": "Missing parameter in body"}, status=400)

```

- Y crear una nueva `Idea`, que se guardará como nueva *fila* en la base de datos:

```

idea = Idea()
idea.title = json_title
idea.description = json_description
idea.category = category
idea.save()
return JsonResponse({"success": True}, status=201)

```

## Pero hay un problema...

¿De quién es la idea?

```

idea = Idea()
idea.user = # ???

```

La respuesta es la misma que...

### ¿...de quién es el *token*?

Vamos a crear una función que devuelve el `usuario` asociado al *token* de la petición, o `None` si hay algún problema:

#### `_get_request_user(request)`

Esta función recibe la `request`, y extrae de las cabeceras el *token de sesión*, como vimos antes:

```

def _get_request_user(request):
    header_token = request.headers.get('Api-Session-Token', None)

```

Si el *token* no está presente en las cabeceras bajo la clave '`Api-Session-Token`' según la especificación, devolvemos `None` directamente:

```

def _get_request_user(request):
    header_token = request.headers.get('Api-Session-Token', None)
    if header_token is None:
        return None

```

En caso contrario, recuperaremos la `UserSession` asociada a dicho *token*. Gracias al *mapeo objeto-relacional*, ¡basta con acceder al atributo `creator` para obtener la *fila* del `User`!

```

def _get_request_user(request):
    header_token = request.headers.get('Api-Session-Token', None)
    if header_token is None:
        return None
    try:
        db_session = UserSession.objects.get(token=header_token)
        return db_session.creator
    except UserSession.DoesNotExist:
        return None

```

## ¡Listos para devolver `401` o seguir con éxito!

Ahora, en nuestra función original:

```
idea = Idea()
authenticated_user = __get_request_user(request)
if authenticated_user is None:
    return JsonResponse({"error": "Not valid token or missing header"}, status=401)
idea.user = authenticated_user
# ...
```

## Terminando

Ya sólo falta modificar `urls.py` para añadir el *mapeo* relevante.

¡Y habremos terminado con éxito nuestra primera petición que requiere **autenticación de usuario**

Puedes verificar cómo funciona lanzando un `cURL` como el de arriba<sup>1</sup>.

## Por último

Verifica que tu código pasa el *test* asociado a la tarea.

Haz `commit` y `push` para subir los cambios al repositorio.

- Si te encuentras con un error `UnicodeDecodeError: 'utf-8' codec can't decode byte`, seguramente se están mandando caracteres especiales (e.g.: `á`, `é`, `í` ...) codificados según su código Unicode. Prueba a usar

`json.loads(request.body.decode('raw_unicode_escape'))` en lugar de `json.loads(request.body)` 



Rubén Montero @ruben.montero changed milestone to %Sprint 5 3 days ago