

# ENUNCIADO PEC 1

## Programación reactiva: Formularios reactivos

**Desarrollo front-end avanzado**

**Máster Universitario en Desarrollo de sitios y aplicaciones web**

UOC

Universitat Oberta  
de Catalunya

### Contenido

- Formato de entrega
- Configuración entorno
- Enunciado
- Puntuación



# Formato de entrega

Se entregará un fichero **zip** con:

- El proyecto **Angular** facilitado en el campus **blog-uoc-project-front** comprimido en un **zip** sin incluir el directorio “**node\_modules**” con la implementación de la solución.
- Un documento de texto (si fuese necesario) comentando cualquier aspecto que sea relevante para tener en cuenta, por ejemplo, algunas decisiones de diseño, alguna parte que no funciona o ha quedado pendiente, ... En definitiva, aspectos que yo como “corrector” deba tener en cuenta.

## Configuración entorno

Lo primero que tendremos que hacer es ver el video del campus **1-ExplicaciónFront**.

En él se explica cómo debería quedar la aplicación una vez terminada. Es decir, se muestra el resultado que deberéis conseguir implementar vosotr@s.

Seguidamente, podéis visualizar los videos **2-ExplicaciónPostman** y **3-Swagger** donde comento como utilizar tanto el **Postman** como el **Swagger**. No obstante, para utilizar estos dos programas primero tenemos que tener la api funcionando. Por lo tanto, comentemos cómo configurar y levantar la API. Aquí hay que añadir que, si estudiamos en su momento la aplicación **MessageApp** de la teoría del tema 1, nos resultará muy sencillo tener la API de esta práctica 1 funcionando.

Básicamente para tener la API funcionando deberíamos:

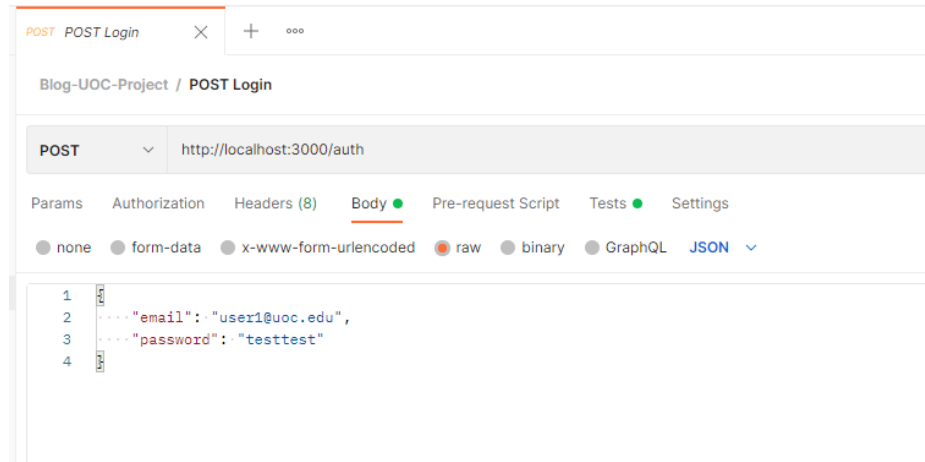
- Crear una base de datos con el **pgAdmin** que se llame **blog-uoc-project**
- Bajar el fichero **4-blog-uoc-project-api.zip**, descomprimirlo y ejecutar los comandos:
  - o **npm install** (instalar todos los paquetes)
  - o **nest start --watch** (arrancar la API, la primera vez nos creará todas las tablas en base de datos, **recordad que primero la tenemos que tener creada**)

Una vez la base de datos creada y la API levantada, podemos configurar el **Postman**.

Podéis volver a ver el video **2-ExplicaciónPostman** para recordar la configuración de la autenticación.

A continuació, os pongo una captura de pantalla de cada llamada que tendréis que utilizar a modo de ayuda:

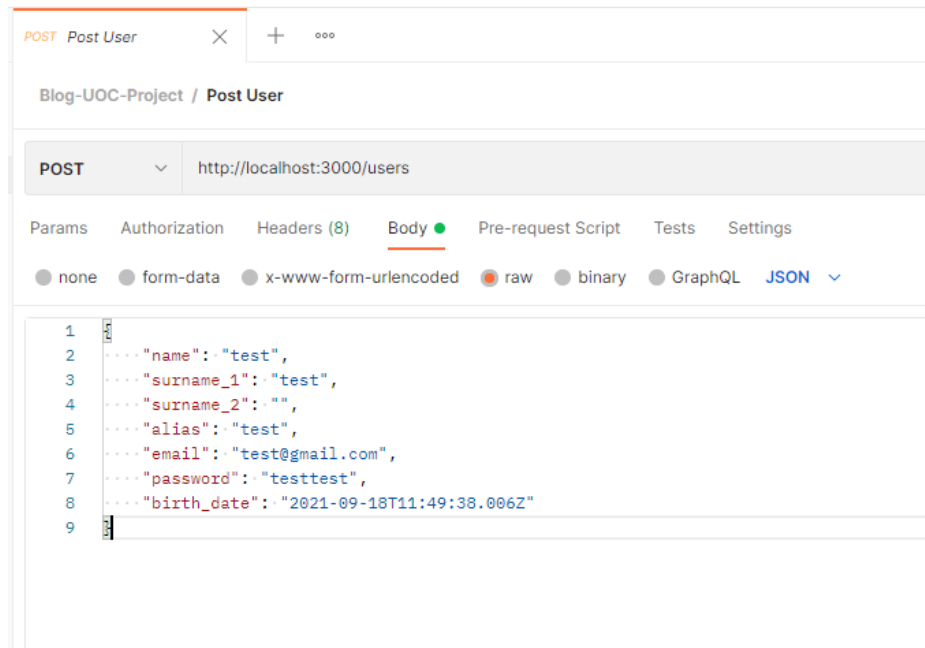
- Autenticación a la plataforma:



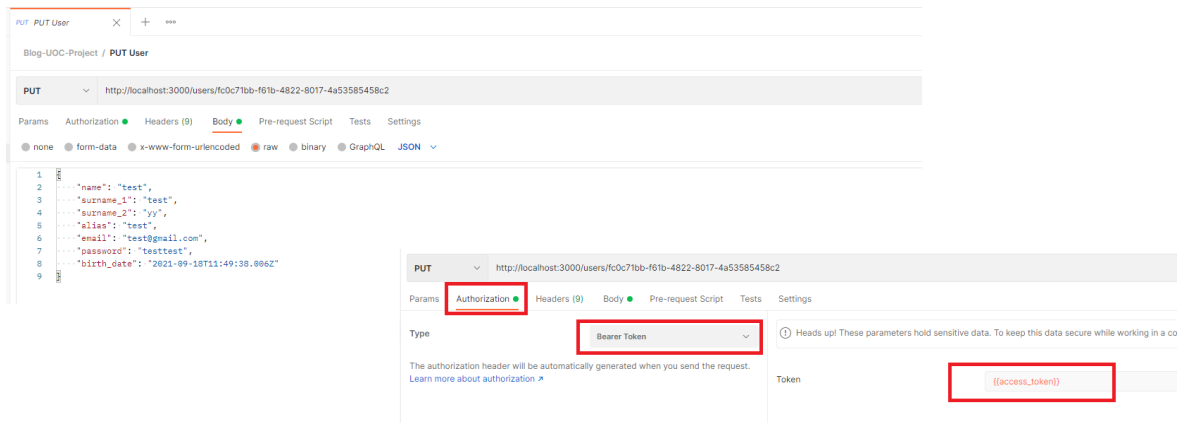
Recordad de poner las siguientes líneas en la pestaña **Tests**:

```
var bodyAuth = JSON.parse(responseBody)
pm.environment.set("access_token", bodyAuth.access_token);
```

- Registro de un usuario nuevo:

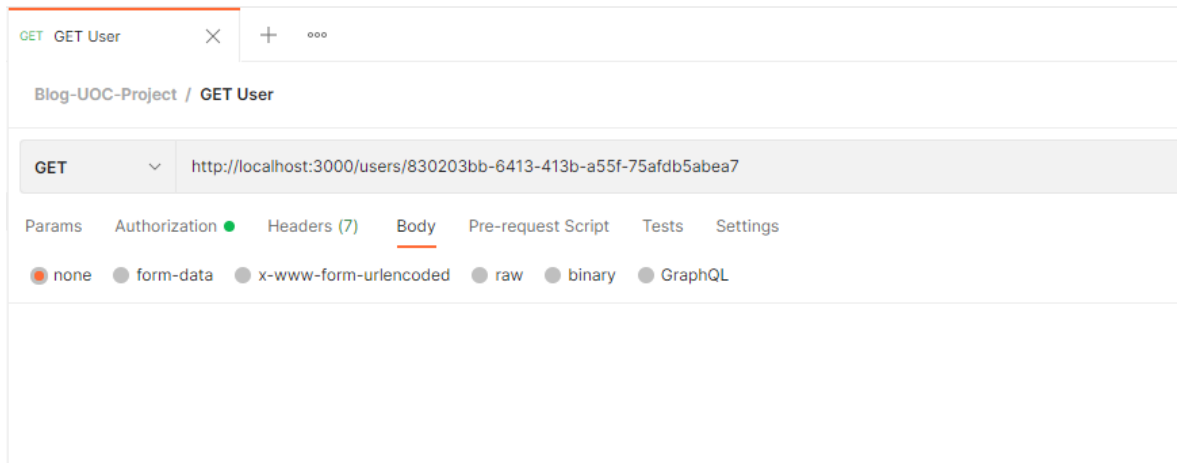


- Actualizar los datos de un usuario:

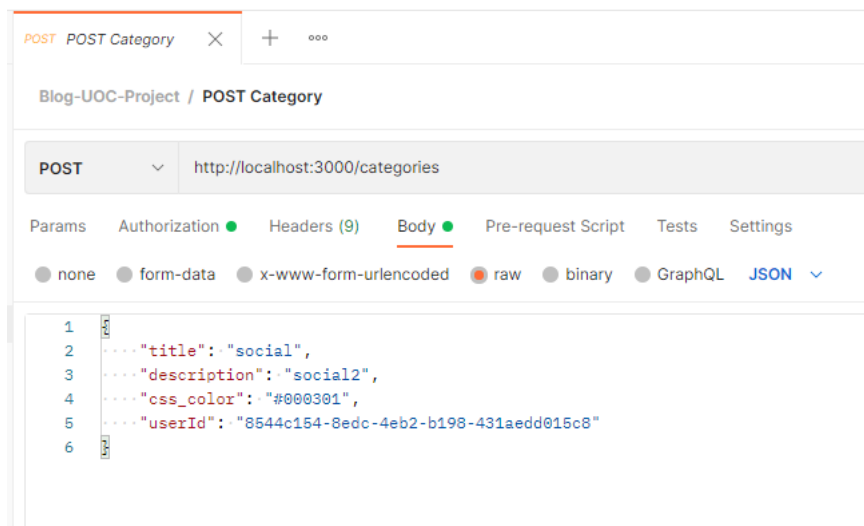


Recordad que todas las llamadas que necesiten autenticación tenemos que ponerle en la pestaña **Authorization** la configuración **Bearer Token** y como token `{{access_token}}`

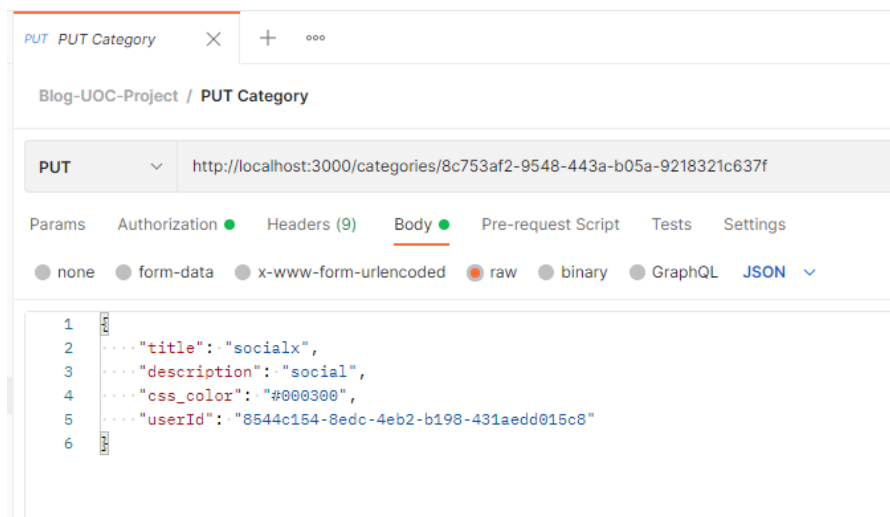
- Devolver datos de usuario según su **id**:



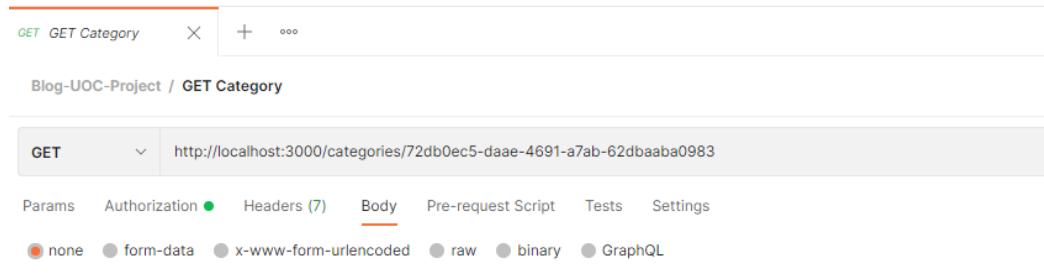
- Dar de alta una categoría:



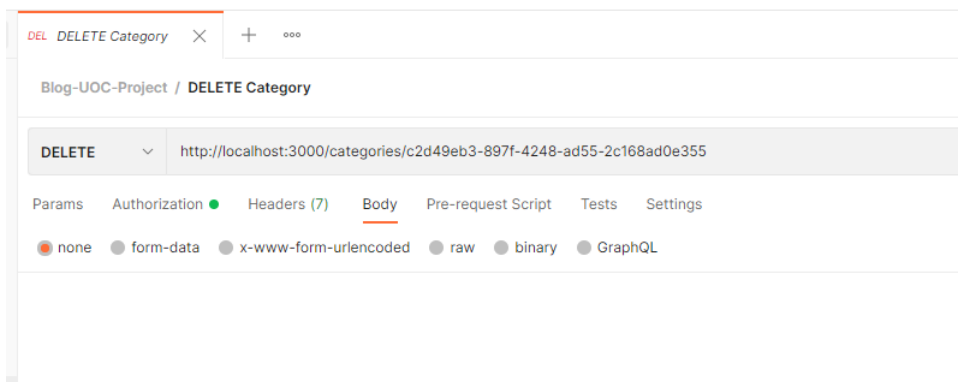
- Actualizar una categoría:



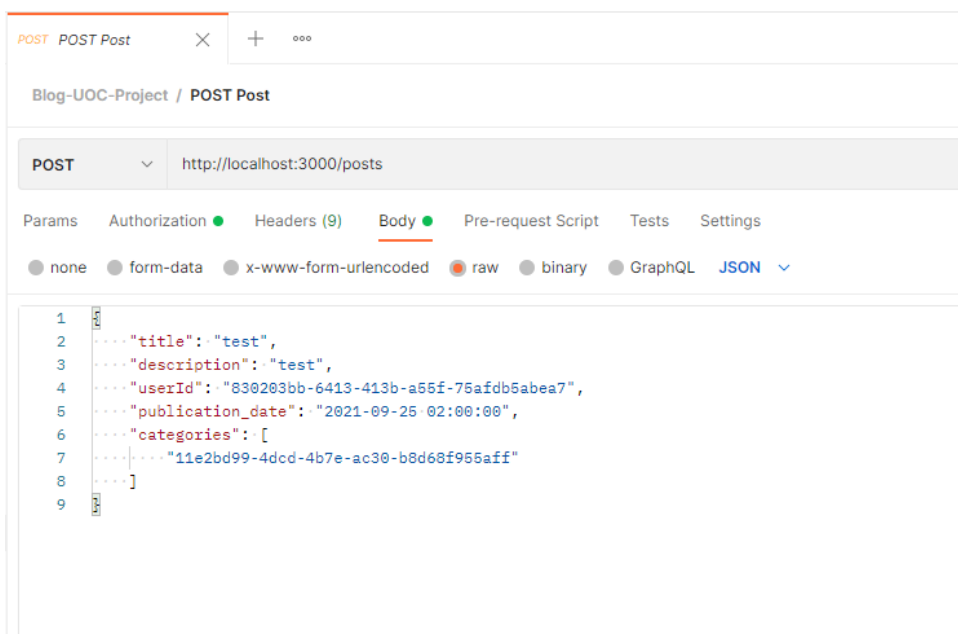
- Devolver una categoría según su **id**:



- Eliminar una categoría:



- Dar de alta un post, fijaros que tenemos que pasarle un array con los **ids** de las categorías asociadas:



- Edición de un post, igual que antes, fijaros que pasamos el array de **ids** de las categorías asociadas:

PUT PUT Post

Blog-UOC-Project / PUT Post

PUT http://localhost:3000/posts/6dd2600b-2549-4954-aa79-a5020cdcfd6

Params Authorization Headers (9) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```

1  {
2    "title": "first post si",
3    "description": "tratra si",
4    "num_likes": 12,
5    "num_dislikes": 3,
6    "userId": "ced861cf-219b-43dc-a17d-d5b83445de16",
7    "categories": [
8      "c04f6300-f221-4667-b13d-94c68e14e050",
9      "41936570-352b-4f81-a9e3-712635b60ddb"
10   ]
11 }
12

```

- Retornar todos los posts:

GET GET Posts

Blog-UOC-Project / GET Posts

GET http://localhost:3000/posts

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

KEY
Key

Body Cookies Headers (8) Test Results

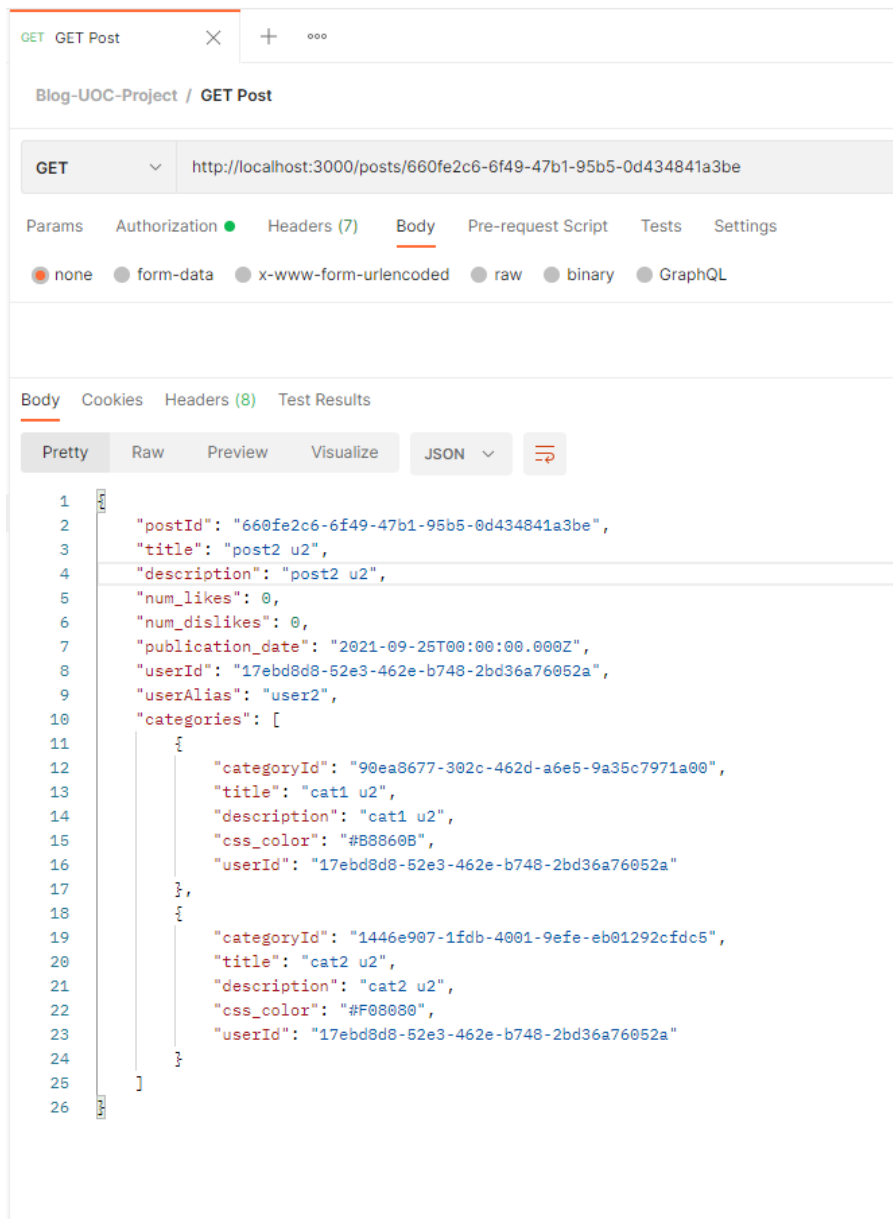
Pretty Raw Preview Visualize JSON

```

1  {
2    "postId": "4c8593c9-0cd1-445c-a892-27b313ff6b2c",
3    "title": "post1",
4    "description": "post1",
5    "num_likes": 5,
6    "num_dislikes": 6,
7    "publication_date": "2021-09-26T00:00:00.000Z",
8    "userId": "830203bb-6413-413b-a55f-75afdb5abea7",
9    "userAlias": "user1",
10   "categories": [
11     {
12       "categoryId": "11e2bd99-4dcd-4b7e-ac30-b8d68f955aff",
13       "title": "cat1",
14       "description": "cat1",
15       "css_color": "#F0F8FF",
16       "userId": "830203bb-6413-413b-a55f-75afdb5abea7"
17     },
18     {
19       "categoryId": "72db0ec5-daae-4691-a7ab-62dbaaba0983",
20       "title": "cat2",
21       "description": "cat2",
22       "css_color": "#FAEBD7",
23       "userId": "830203bb-6413-413b-a55f-75afdb5abea7"
24     }
25   ]
26 },
27 {
28   "postId": "86e04b4d-173f-4098-ae3f-06b7b8b69210",
29   "title": "post2",
30   "description": "post2",
31   "num_likes": 5,
32   "num_dislikes": 4,
33   "publication_date": "2021-09-26T00:00:00.000Z",
34   "userId": "830203bb-6413-413b-a55f-75afdb5abea7",
35   "userAlias": "user1",
36   "categories": [
37     {
38       "categoryId": "11e2bd99-4dcd-4b7e-ac30-b8d68f955aff",
39       "title": "cat1",
40       "description": "cat1",
41       "css_color": "#F0F8FF",
42       "userId": "830203bb-6413-413b-a55f-75afdb5abea7"
43     },
44     {
45       "categoryId": "72db0ec5-daae-4691-a7ab-62dbaaba0983",
46       "title": "cat2",
47       "description": "cat2",
48       "css_color": "#FAEBD7",
49       "userId": "830203bb-6413-413b-a55f-75afdb5abea7"
50     }
51   ]
52 }
53 ]
54

```

- Retornar un post según su id:



GET GET Post

Blog-UOC-Project / GET Post

GET http://localhost:3000/posts/660fe2c6-6f49-47b1-95b5-0d434841a3be

Params Authorization Headers (7) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

Body Cookies Headers (8) Test Results

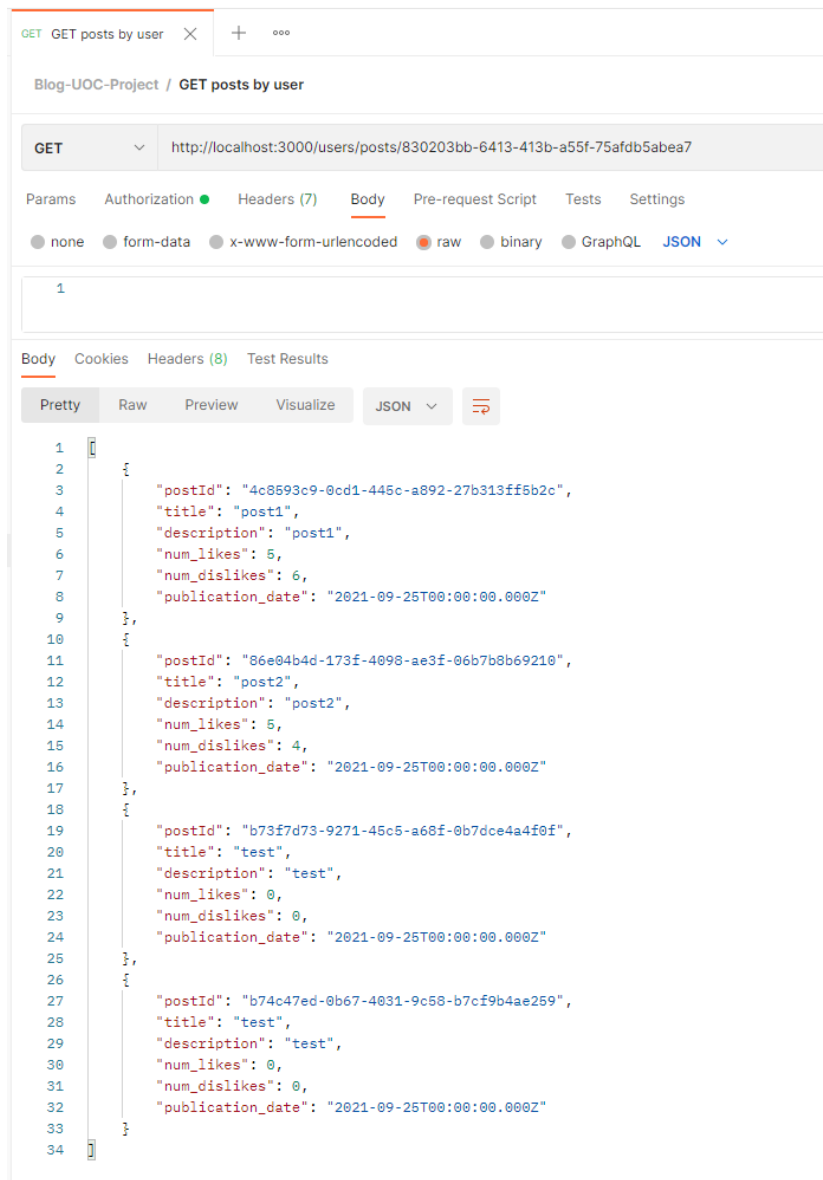
Pretty Raw Preview Visualize JSON

```

1  {
2    "postId": "660fe2c6-6f49-47b1-95b5-0d434841a3be",
3    "title": "post2 u2",
4    "description": "post2 u2",
5    "num_likes": 0,
6    "num_dislikes": 0,
7    "publication_date": "2021-09-25T00:00:00.000Z",
8    "userId": "17ebd8d8-52e3-462e-b748-2bd36a76052a",
9    "userAlias": "user2",
10   "categories": [
11     {
12       "categoryId": "90ea8677-302c-462d-a6e5-9a35c7971a00",
13       "title": "cat1 u2",
14       "description": "cat1 u2",
15       "css_color": "#B8860B",
16       "userId": "17ebd8d8-52e3-462e-b748-2bd36a76052a"
17     },
18     {
19       "categoryId": "1446e907-1fdb-4001-9efe-eb01292cfdc5",
20       "title": "cat2 u2",
21       "description": "cat2 u2",
22       "css_color": "#F08080",
23       "userId": "17ebd8d8-52e3-462e-b748-2bd36a76052a"
24     }
25   ]
26 }
```



- Retornar todos los posts de un usuario concreto:



GET GET posts by user

Blog-UOC-Project / GET posts by user

GET http://localhost:3000/users/posts/830203bb-6413-413b-a55f-75afdb5abea7

Params Authorization Headers (7) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

1

Body Cookies Headers (8) Test Results

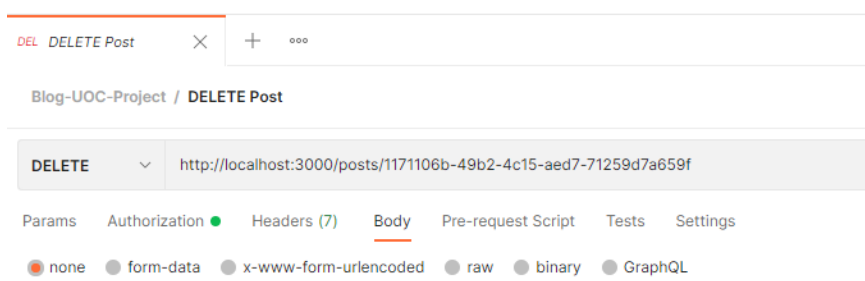
Pretty Raw Preview Visualize JSON

```

1  {
2    {
3      "postId": "4c8593c9-0cd1-445c-a892-27b313ff5b2c",
4      "title": "post1",
5      "description": "post1",
6      "num_likes": 5,
7      "num_dislikes": 6,
8      "publication_date": "2021-09-25T00:00:00.000Z"
9    },
10   {
11     "postId": "86e04b4d-173f-4098-ae3f-06b7b8b69210",
12     "title": "post2",
13     "description": "post2",
14     "num_likes": 5,
15     "num_dislikes": 4,
16     "publication_date": "2021-09-25T00:00:00.000Z"
17   },
18   {
19     "postId": "b73f7d73-9271-45c5-a68f-0b7dce4a4f0f",
20     "title": "test",
21     "description": "test",
22     "num_likes": 0,
23     "num_dislikes": 0,
24     "publication_date": "2021-09-25T00:00:00.000Z"
25   },
26   {
27     "postId": "b74c47ed-0b67-4031-9c58-b7cf9b4ae259",
28     "title": "test",
29     "description": "test",
30     "num_likes": 0,
31     "num_dislikes": 0,
32     "publication_date": "2021-09-25T00:00:00.000Z"
33   }
34 }

```

- Eliminar un post:



DEL DELETE Post

Blog-UOC-Project / DELETE Post

DELETE http://localhost:3000/posts/1171106b-49b2-4c15-aed7-71259d7a659f

Params Authorization Headers (7) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL

- Devolver las categorías de un usuario:

GET GET categories by... X + ...

Blog-UOC-Project / GET categories by user

GET http://localhost:3000/users/categories/830203bb-6413-413b-a55f-75afdb5abea7

Params Authorization Headers (7) Body Pre-request Script Tests Settings

Type Bearer Token

The authorization header will be automatically generated when you send the request.  
[Learn more about authorization](#)

Heads up! These parameters are required for this authentication type.

Token

Body Cookies Headers (8) Test Results

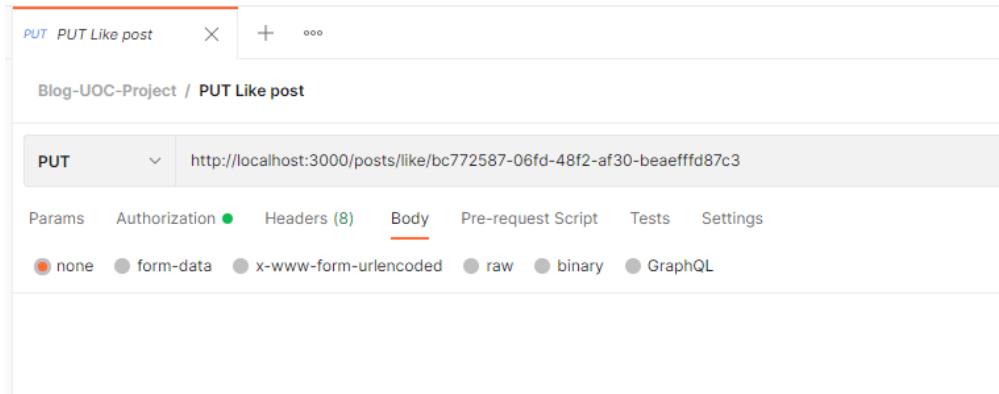
Pretty Raw Preview Visualize JSON

```

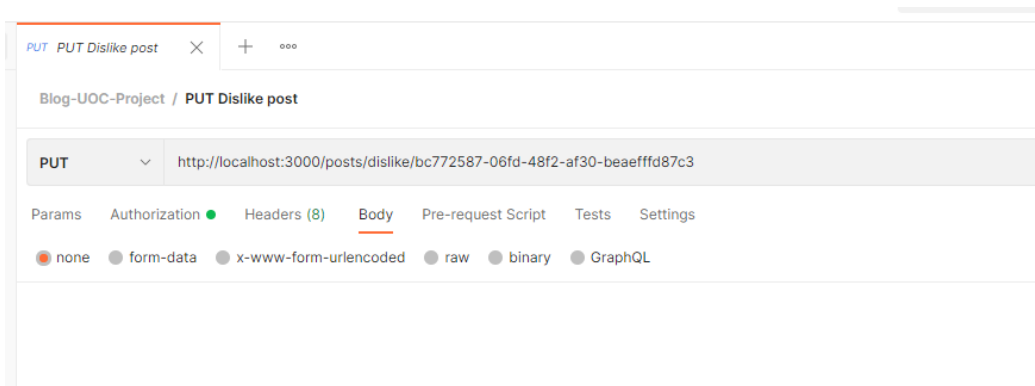
1  {
2    {
3      "categoryId": "11e2bd99-4dcd-4b7e-ac30-b8d68f955aff",
4      "title": "cat1",
5      "description": "cat1",
6      "css_color": "#F0F8FF"
7    },
8    {
9      "categoryId": "72db0ec5-daae-4691-a7ab-62dbaaba0983",
10     "title": "cat2",
11     "description": "cat2",
12     "css_color": "#FAEBD7"
13   },
14   {
15     "categoryId": "fa83e721-1009-4457-aadf-fdce1d0b456c",
16     "title": "cat3",
17     "description": "cat3",
18     "css_color": "#A52A2A"
19   }
20 }

```

- Enviar un like:



- Enviar un dislike:



Una vez tengamos el **Postman** configurado, os recomendamos interactuar con la API.

Haced un pequeño juego de pruebas igual que os hago en el video donde os enseño el resultado de la web, pero desde **Postman**. De esta manera, os acostumbráis a utilizar las llamadas usando esta herramienta, los **body** que tenéis que enviar en algunas peticiones y sobre todo, las respuestas que recibes, las cuales tendréis que tratar en el **frontend**.

Una vez realizadas las pruebas, podéis limpiaros la base de datos ejecutando las instrucciones **SQL** siguientes:

- `delete from public.categories;`
- `delete from public.posts;`
- `delete from public.users;`

Para lanzar sentencias **SQL** en el **pgAdmin** podéis hacer clic derecho con el ratón sobre el nombre de la base de datos **blog-uoc-project** y seleccionar la opción **Query Tool**. Se os abrirá una pestaña donde podéis escribir vuestras sentencias **SQL**. Es importante desenvolverse de manera básica en cuanto a **SQL** se refiere.

Una vez familiarizados con todas los **endpoints**, sería momento de levantar el **frontend** e implementar la práctica propiamente dicha.

Solo tendríamos que bajarnos el fichero del campus **blog-uoc-project-front.zip**, descomprimirlo en la carpeta que toque y ejecutar desde la consola los comandos:

- **npm install**
- **ng serve**

# Enunciado

Cuando empecéis con el desarrollo de esta práctica, dos cuestiones para tener en cuenta:

- Por una parte, intentad resolver los ejercicios en el orden propuesto, la práctica está pensada para que sea progresiva.
- Por otra parte, revisad todo lo que tenéis implementado, sobre todo:
  - Las rutas definidas, las **guards** implementadas. Fijaros que la **guard** que valida si existe el **access\_token** en el **localStorage**, si no existe redirige al usuario a la pantalla de **login**.
  - Los diferentes servicios implementados de las diferentes entidades **user**, **category**, **post** y **auth**. Utilizamos promesas en esta primera práctica, en la práctica 2 aparte de aplicar **Redux** pasaremos las llamadas **http** a **observables** y así tendremos las dos opciones estudiadas.
  - Servicios auxiliares como:
    - **auth-interceptor** que nos añade en los **headers** de todas las llamadas el **access\_token** para poder utilizar los **endpoints** protegidos de la API.
    - **header-menus** es interesante ya que mediante un **BehaviorSubject** podemos manejar los puntos de menú para cuando estemos autenticados o no gracias a que un componente envía el dato que sea mediante el método **next** y el componente **header** en este caso tiene el método **subscribe**, de manera que mediante variables booleanas se puede gestionar cuando mostrar los menús de zona pública o zona privada.
    - **local-storage** simplemente para tener encapsulado en un servicio los métodos de acceso al **localStorage**.
    - **shared** para gestionar el **toast** para darnos **feedback** si la respuesta es correcta o no por parte de la API.

## Ejercicio 1

En el primer ejercicio vamos a terminar la implementación del registro. Cuando vayamos al punto de menú **Register**, deberemos insertar nuestros datos de usuario y darnos de alta en la plataforma. Para ello, deberemos completar la implementación del componente **register.component**, concretamente:

En el controlador **register.component.ts**:

- Descomentar la declaración de variables

Buscad en el proyecto la etiqueta:

### TODO 16

- En el **constructor** deberemos:
  - Inicializar el objeto **registerUser**.
  - Inicializar la variable **isValidForm** a **null**.
  - Inicializar las diferentes propiedades del formulario:
    - **name**: requerido, mínimo 5 / máximo 25 caracteres.
    - **surname\_1**: requerido, mínimo 5 / máximo 25 caracteres.
    - **surname\_2**: opcional, mínimo 5 / máximo 25 caracteres.
    - **alias**: requerido, mínimo 5 / máximo 25 caracteres.
    - **birth\_date**: requerido, formato 'yyyy-mm-dd'.
    - **email**: requerido, formato de **email** válido.
    - **password**: requerido, mínimo 8 caracteres, máximo 16.
    - **registerForm**: definir el **formBuilder** con todas las propiedades anteriores.

Buscad en el proyecto la etiqueta:

### TODO 17

En la vista **register.component.html**:

- Implementar el formulario reactivo para gestionar todas las propiedades anteriores. El formulario al hacer **submit** tendrá que llamar al método **register** que tenéis implementado en el controlador, solo debéis descomentar el código para poder utilizarlo.

Buscad en el proyecto la etiqueta:

### TODO 18

- No quitéis el **div** de arriba del todo, vincula el **id="registerFeedback"** con los estilos del propio componente y con la gestión del **toast** cuando se llama al **this.sharedService.managementToast(...)** para gestionar la respuesta ok o ko de la API.

## Ejercicio 2

En el segundo ejercicio vamos a terminar la implementación de la autenticación en el sistema. Cuando vayamos al punto de menú **Login**, deberemos poder insertar nuestras credenciales y autenticarnos en la plataforma. Para ello, deberemos completar la implementación del componente **login.component**, concretamente:

En el controlador **login.component.ts**:

- Descomentar la declaración de variables.

Buscad en el proyecto la etiqueta:

### TODO 19

- En el **constructor** deberemos:
  - Inicializar el objeto **loginUser**.
  - Inicializar las diferentes propiedades del formulario:
    - **email**: requerido, formato de **email** válido.
    - **password**: requerido, mínimo 8 caracteres, máximo 16.
    - **loginForm**: definir el **formBuilder** con todas las propiedades anteriores.

Buscad en el proyecto la etiqueta:

### TODO 20

En la vista **login.component.html**:

- Implementar el formulario reactivo para gestionar todas las propiedades anteriores. El formulario al hacer **submit** tendrá que llamar al método **login** que tenéis implementado en el controlador, solo debéis descomentar el código para poder utilizarlo.

Buscad en el proyecto la etiqueta:

### TODO 21

- No quitéis el **div** de arriba del todo, vincula el **id="loginFeedback"** con los estilos del propio componente y con la gestión del **toast** cuando se llama al **this.sharedService.managementToast(...)** para gestionar la respuesta ok o ko de la API.

### Ejercicio 3

En este ejercicio vamos a implementar el **logout**.

En el fichero **header.component.ts** en el método **logout** tenemos que implementar la siguiente lógica (**TODO 15**):

- Utilizando el método **remove** del servicio **localStorageService** eliminar el campo **user\_id**
- Utilizando el método **remove** del servicio **localStorageService** eliminar el campo **access\_token**
- Crear una constante **headerInfo** de tipo **HeaderMenus** con las propiedades:
  - **showAuthSection: false**
  - **showNoAuthSection: true**
- Añadir la siguiente línea para actualizar los puntos de menú:

```
this.headerMenusService.headerManagement.next(headerInfo);
```

- Y finalmente redireccionar a la **home**



## Ejercicio 4

Cuando estemos autenticados al sistema y vayamos al punto de menú **Profile**, deberemos poder editar nuestros datos de usuario. Para ello, deberemos completar la implementación del componente **profile.component**, concretamente:

En el controlador **profile.component.ts**:

- Descomentar la declaración de variables.

Buscad en el proyecto la etiqueta:

### TODO 4

- En el **constructor** deberemos:
  - Inicializar el objeto **profileUser**.
  - Inicializar la variable **isValidForm** a **null**.
  - Inicializar las diferentes propiedades del formulario:
    - **name**: requerido, mínimo 5 / máximo 25 caracteres.
    - **surname\_1**: requerido, mínimo 5 / máximo 25 caracteres.
    - **surname\_2**: opcional, mínimo 5 / máximo 25 caracteres.
    - **alias**: requerido, mínimo 5 / máximo 25 caracteres.
    - **birth\_date**: requerido, formato 'yyyy-mm-dd'.
    - **email**: requerido, formato de **email** válido.
    - **password**: requerido, mínimo 8 caracteres, máximo 16.
    - **profileForm**: definir el **formBuilder** con todas las propiedades anteriores.

Buscad en el proyecto la etiqueta:

### TODO 5

En la vista **profile.component.html**:

- Implementar el formulario reactivo para gestionar todas las propiedades anteriores. El formulario al hacer **submit** tendrá que llamar al método **updateUser** que tenéis implementado en el controlador, solo debéis descomentar el código para poder utilizarlo.

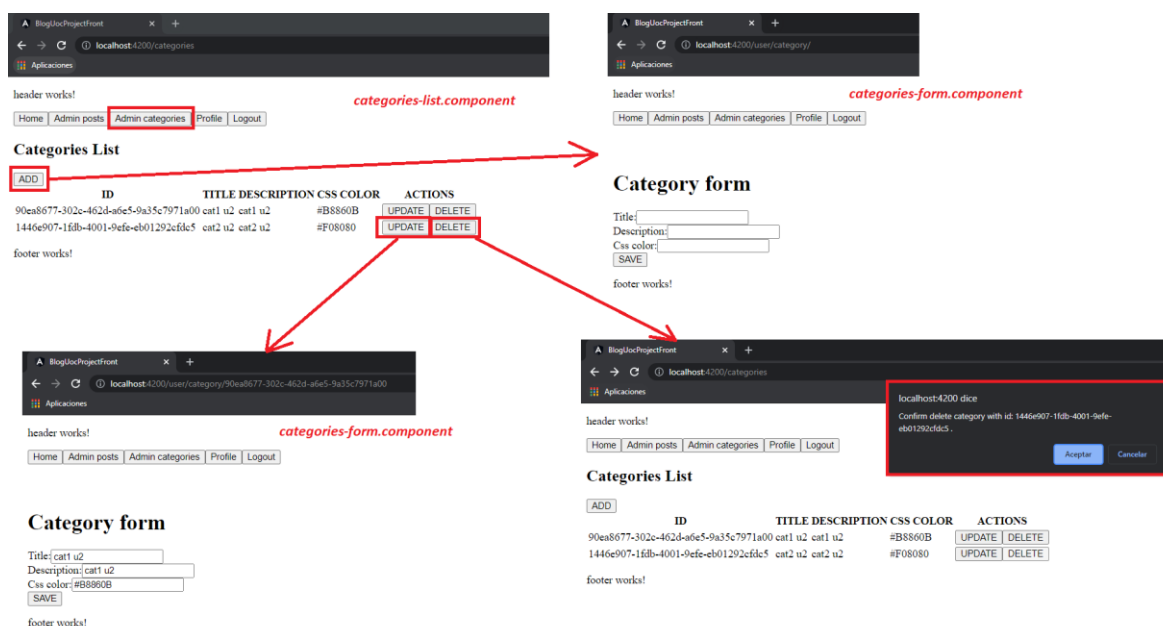
Buscad en el proyecto la etiqueta:

## TODO 6

- No quitéis el **div** de arriba del todo, vincula el **id="profileFeedback"** con los estilos del propio componente y con la gestión del **toast** cuando se llama al **this.sharedService.managementToast(...)** para gestionar la respuesta ok o ko de la API.

## Ejercicio 5

En este ejercicio vamos a terminar de implementar el CRUD de la entidad categorías. Podemos volver a visual el video para ver como interactuar con esta entidad. A modo de resumen, podemos observar la siguiente imagen:



Proponemos los siguientes pasos:

- Estudia el componente **categories-list**, tanto la vista como el controlador.
- Observa como en el controlador cargamos todas las categorías en función del usuario.
- Si hacemos clic al botón de crear una nueva categoría redirigiremos al componente **categories-form** sin pasar un **categoryId**, añade la siguiente línea en el método **createCategory** (TODO 7):

```
this.router.navigateByUrl('/user/category/');
```

- Si hacemos clic al botón de editar una categoría redirigiremos al componente **categories-form** pasando un **categoryId** por **url**, añade la siguiente línea en el método **updateCategory** (TODO 8):

```
this.router.navigateByUrl('/user/category/' + categoryId);
```

- Si hacemos clic al botón de eliminar una categoría mostraremos un mensaje de confirmación, si pulsamos **cancelar** no se realiza ninguna acción. En cambio, si pulsamos **aceptar** se elimina la categoría de la base de datos y tendremos que recargar el listado de categorías de la página actual, por lo tanto, añade la siguiente línea en el método **deleteCategory**(**TODO 9**):

```
this.loadCategories();
```

### Valida el comportamiento anterior.

- Vamos a estudiar el componente **categories-form**. En este componente podemos llegar tanto si es un alta como si es una edición de una categoría.
- Estudia el **constructor**, fijate en cómo se inicializan las diferentes variables y los diferentes campos del formulario.
- Estudia el **ngOnInit**, en él gestionamos si se trata de un alta o de una edición mediante el campo **categoryId**. Recordamos que lo enviamos por **url** si es una edición, si es un alta no. En caso de ser una edición, pedimos los datos de la categoría en función de su **id** para cargarlos en el formulario.
- Estudia las dos funciones de editar y crear una categoría. Fíjate cómo leemos el identificador del usuario del **localStorage** y cómo hacemos la llamada al servicio de actualizar o dar de alta en función de lo requerido. Fijaros por último cómo utilizamos el servicio **managementToast** para mostrar el feedback al usuario de si la respuesta ha ido bien o no.
- Finalmente, en el método **saveCategory** nos falta implementar una pequeña lógica:
  - Sustituir el **TODO 10** por:
  - Implementar un **if / else** de manera que:
  - Si la variable **isUpdateMode** es **true**:

```
this.validRequest = await this.editCategory();
```

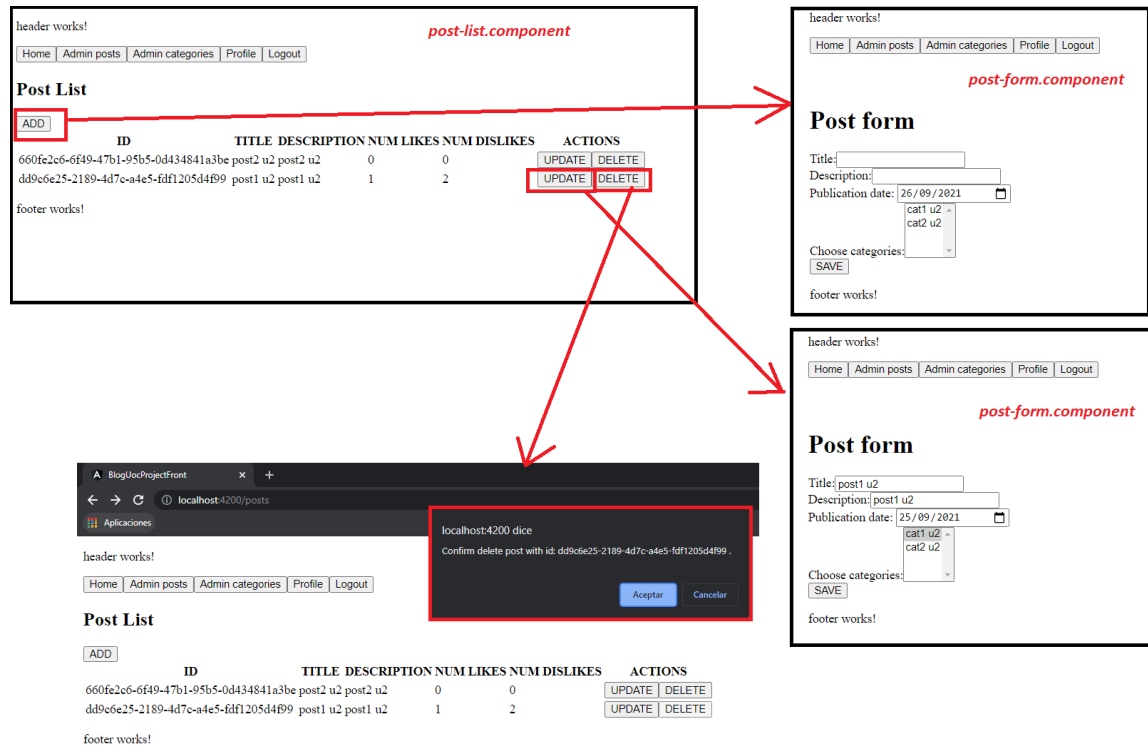
- Si no:

```
this.validRequest = await this.createCategory();
```

## Ejercicio 6

En este ejercicio implementaremos el CRUD de la entidad posts. Este CRUD tendréis que implementarlo todo vosotr@s desde cero. Para ello, **podemos ayudarnos del CRUD de categorías anteriores ya que es lo mismo en gran parte.**

El resultado es el siguiente:



En el componente **post-list.component** (**TODO 11** y **TODO 12**):

En la vista **post-list.component.html** tendremos que implementar:

- El botón de añadir post.
- El listado de posts con un **ngFor** que muestre los campos:
  - postId.
  - title.
  - description.
  - num\_likes.
  - num\_dislikes.
  - Por cada fila las acciones de actualizar y eliminar

En el controlador **post-list.component.ts** tendremos que implementar:

- Una variable **posts!**: **PostDTO[]**
- La carga de todos los posts del usuario mediante el servicio **getPostsByUserId**
  - o Tendréis que implementar el servicio **getPostsByUserId** del fichero **post.service.ts** ( **TODO 22** ).
- El método **createPost** que redirigirá a **'user/post/'** sin pasar un **postId**.
- El método **updatePost** que redirigirá a **'user/post/' + postId**.
- El método **deletePost** que pedirá confirmación para eliminar el post. Si se cancela la confirmación no se realizará ninguna acción y si se confirma se llamará al servicio **deletePost** y se volverá a cargar el listado de post para que no salga el último eliminado.

En el componente **post-form.component** (**TODO 13 y TODO 14**):

En el controlador **post-form.component.ts** tendremos que implementar toda la lógica al igual que en el CRUD de categoría para diferenciar entre alta y edición y hacer uso del **managementToast** para mostrar **feedback** al usuario.

- Respecto a las propiedades del formulario de la entidad **post**:
  - o **title**: requerido, máximo 55 caracteres.
  - o **description**: requerido, máximo 255 caracteres.
  - o **publication\_date**: requerido, formato 'yyyy-mm-dd'.
  - o **categories**: array de categorías, tendrá que ser un **desplegable con la opción de selección múltiple**.
- En el controlador necesitaremos implementar:
  - o **loadCategories**: para cargar las categorías del usuario.
  - o **ngOnInit**: gestionaremos el caso de si es una edición, para cargar los datos del formulario.
  - o **editPost**: para lanzar el servicio **updatePost** (tendremos que aplicar la misma lógica que en el CRUD de categorías).
  - o **createPost**: para lanzar el servicio **createPost** (tendremos que aplicar la misma lógica que en el CRUD de categorías).
  - o **savePost**: si el formulario es válido en función de si es alta o edición llamar a **editPost** o **createPost**.

En la vista **post-form.component.html** tendremos que implementar:

- Implementar el formulario reactivo para gestionar todas las propiedades de la entidad **post** (**title**, **description**, **publication\_date** y **categories** asociadas) El formulario al hacer **submit** tendrá que llamar al método **savePost**.
- No quitéis el **div** de la parte superior ya que vincula el **id="postFeedback"** con los estilos y con el servicio **managementToast**. Tendréis que hacer uso de manera semejante a como se utiliza en el CRUD de categorías.

## Ejercicio 7

En el componente **home.component** tendremos el listado de todos los posts de la plataforma. El componente está por terminar. La idea sería que se muestre el listado de posts más o menos como se muestra en el video. Maquetad el aspecto como queráis. Cosas que debemos tener en cuenta:

- En el controlador **home.component.ts**: falta implementar la carga de todos los posts en el método **loadPosts**. Para ello debemos seguir la lógica siguiente:
  - Definir la variable **errorResponse**.
  - Leer de **localStorageService** el campo **userId**, si lo tenemos, asignar a la variable **showButtons** a **true**. Esta variable nos servirá después para mostrar o no los botones de **like** o **dislike** de la vista.
  - Implementar el **try/catch** para llamar al **getPosts** y guardar el resultado en la variable **posts**.
  - En caso de error pasárselo al **sharedService.errorLog**.

Buscad en el proyecto la etiqueta:

### TODO 2

- Implementar la vista **home.component.html**: en la vista tendremos que implementar un **ngFor** de la variable **posts** y maquetar cada post. En cada post mostraremos todos los campos y además todas las categorías asociadas. Pensad que a cada categoría tenemos que pasarle en los estilos su color de fondo que viene definido por la propiedad **category.css\_color**. Finalmente, en función de la variable **showButtons** mostraremos los botones de **like** y **dislike**.

Buscad en el proyecto la etiqueta:

### TODO 3

**Fijaros como queda el resultado en el video explicativo para tener una referencia.**

## Ejercicio 8

En la pantalla **home**, donde tenemos el listado de todos los posts de la plataforma, de entre todos los campos de un post tenemos el campo fecha de publicación. Si mostramos el campo tal cual nos viene de base de datos veríamos algo así:



En este ejercicio se pide implementar un **pipe** de manera que de un formato amigable a la fecha. Concretamente deberemos implementar un **pipe** que reciba un numero como argumento, de manera que:

- Pasamos un 1 nos devolverá el formato `25092021`
- Pasamos un 2 nos devolverá el formato `25 / 09 / 2021`
- Pasamos un 3 nos devolverá el formato `25/09/2021`
- Pasamos un 4 nos devolverá el formato `2021-09-25`

*\*\* Pensad que si el número del día o del mes es menor a 10 tendremos que concatenarle un zero delante nosotr@s.*

Buscad en el proyecto la etiqueta:

### TODO 1

**Fijaros como queda el resultado en el video explicativo para tener una referencia.**

## Ejercicio 9

Una vez realizada toda la implementación anterior y tengamos la plataforma funcionando, vamos a implementar un último ejercicio obligatorio. Los requisitos son los siguientes:

- Implementar una nueva ruta **dashboard** accesible desde un punto de menú **DASHBOARD** de tipo público, es decir, no será necesario estar autenticado al sistema para poder acceder a la vista.

- Implementar el componente **dashboard** al que navegaremos desde la nueva ruta anteriormente comentada. En este componente necesitaremos:
  - Consumir el **endpoint** que nos devuelve todos los **posts** de la plataforma (el mismo utilizado en el componente **home**)
  - En el controlador del componente **dashboard**, recorrer los posts que nos devuelve el **endpoint** anterior y calcular el número total de **likes** y el número total de **dislikes** y mostrar estos dos datos en la vista
    - El cálculo deberá hacerse en el controlador, asignarse a una variable pública y ésta será la que utilizaremos en la vista para mostrar los dos datos.

### Ejercicio 10 (OPCIONAL)

La idea sería que una vez terminada toda la implementación, si queremos ir un puntito más allá, investiguemos sobre estas dos propuestas:

- Por una parte, en la vista **home**, donde tenemos el listado de todos los posts de la plataforma, podríamos implementar algún tipo de filtro en la parte superior del listado. Por ejemplo, un filtro por fecha, un filtro por texto y que busque el texto si está incluido en los títulos o las descripciones de los posts, un filtro por alias, ... En definitiva, intentar aplicar algún filtro. Pensad que esto sería a nivel de **frontend**, es decir, con la información que tenéis en el **frontend** deberías poder resolver varios ejemplos de filtros. No hay **endpoints** específicos para ello.
- Por otra parte, otro ejercicio que podríais investigar es el de aplicar el paquete **i18n**, es decir, el multiidioma. Básicamente la idea sería instalar el paquete y poner todas las traducciones en un fichero **es.json** y **en.json** y sustituir los textos **hardcode** de las vistas por los campos correspondientes del **i18n**. Una vez esto configurado, podríamos implementar un desplegable en la parte superior de la aplicación donde poder cambiar entre los idiomas **es** y **en**.



# Puntuación

A continuación, mostramos cuánto puntúan cada uno de los ejercicios de la práctica para obtener la nota final de la misma. Hay que tener en cuenta que la práctica se puntúa sobre 10, y tenemos un punto extra si implementamos el ejercicio 10 que es opcional.

- Ejercicio 1 [ **1 punto** ]
- Ejercicio 2 [ **1 punto** ]
- Ejercicio 3 [ **0,5 puntos** ]
- Ejercicio 4 [ **1 punto** ]
- Ejercicio 5 [ **1 punto** ]
- Ejercicio 6 [ **3 puntos** ]
- Ejercicio 7 [ **1 punto** ]
- Ejercicio 8 [ **1 punto** ]
- Ejercicio 9 [ **0,5 punto** ]
- Ejercicio 10 [ **1 punto** ]