# MEMORIA DEL PROYECTO FINAL 2022 DE TDW

# Introducción

El objetivo de este proyecto es crear una aplicación web utilizando otras prácticas realizadas en la asignatura de TDW que permita a un usuario realizar la gestión de aportaciones a la ciencia con productos, entidades y personas.

La aplicación sigue un patrón arquitectónico cliente servidor y en la práctica se hace uso del ORM(mapeador relacional de objetos) doctrine, que hace uso del patrón de diseño Data Mapper.

Este trabajo será realizado con html y css (del ejercicio del tema HTML), javascript (de la Práctica de Front-End), en la parte de frontend se distinguen dos subsistemas diferenciados según el tipo de usuario (reader o writer) que se identifique mediante un nombre de usuario y una contraseña.

El subsistema tipo writer permite operar a un usuario writer con las mismas opciones que había en los casos de uso de la primera práctica a través de un menú. Se ha añadido una opción para la gestión completa (consultas, modificaciones y bajas) de todos los datos del sistema (usuarios, productos, entidades y personas guardadas por los usuarios). Pueden permitir el acceso al sistema a los usuarios registrados (pudiéndose asignar permisos de reader o de writer). Al dar de baja a un usuario, se eliminan todos sus datos correspondientes(delete cascade). Hemos añadido una funcionalidad que permite que un writer pueda suspender a un usuario cambiando su estado actividad a "inactivo" hasta que un writer se lo cambie a "activo". Este método permite conservar los datos de un usuario suspendido para que éste pueda recuperar el acceso a ellos más tarde

El subsistema tipo reader permite a un usuario no registrado acceder a la pantalla de presentación y "registrarse" con un nombre de usuario y una contraseña nuevos para nuestro sistema, a continuación el usuario accederá a una página en la que podrá introducir el resto de sus datos personales (correo electrónico, fecha de nacimiento, urls, etc.). Será necesario que un usuario writer lo autorice para que el usuario nuevo pueda acceder a las diferentes funcionalidades del sistema, que consisten en las opciones contempladas en los casos de uso indicados en la primera práctica.

El usuario reader dispone de una opción para gestionar su cuenta personal ("perfil") pudiendo ver y/o modificar sus datos personales (nombre, dirección de correo electrónico, fecha de nacimiento y contraseña). El identificador de usuario que se ha introducido al registrarse es único en el sistema y se permite su visualización pero no su modificación, además éste está presente en todo momento desde que el usuario ha accedido.

El backend se ha realizado con PHP ampliando e integrando la API REST realizada en el trabajo de php, que incorpora los usuarios del sistema.

También se ha dejado de utilizar el localstorage del navegador que se utilizaba en la práctica de javascript y se ha reemplazado por la base de datos de la práctica de php que nos permite tener persistencia en los datos.

# Instalación:

Para comenzar el proyecto abrimos un terminal del ordenador el directorio raíz del proyecto y ejecutamos el comando:

> composer install

Composer es un estándar para administrar, descargar e instalar dependencias y librerías. Este proyecto tiene múltiples fuentes de instalación, el composer descarga cada dependencia automáticamente, así no hay que hacerlo manualmente.

Doctrine es un Object-Relational Mapper que proporciona persistencia transparente para objetos PHP a través del patrón Data Mapper con el objetivo de obtener un desacoplamiento completo entre la lógica de negocio y la persistencia de los datos en los sistemas de gestión de bases de datos.

> bin\doctrine orm:clear-cache:metadata

Este comando sirve para limpiar la memoria caché de los metadatos (se explica en el problema 1 de la sección problemas encontrados en este documento.).

Repetimos la instalación del composer:

> composer install

Para la ejecución del proyecto se utiliza xampp, que nos proporciona un servidor web con apache, que es el encargado de gestionar bases de datos con un MySQL, una vez instalado el proyecto se accederá a la aplicación en la ruta <a href="http://127.0.0.1:8000/">http://127.0.0.1:8000/</a> (direccion ip y puerto 8000).

Una vez esté instalado se creará un usuario tipo writer y a continuación se debe poner como inactivo al usuario adminUser.

# Cambios en PHP:

Para comenzar esta práctica hemos añadido en la clase "User" los atributos "birthDate", "Url", "name" y "Activo":

```
/**

* @ORM\Column(

* name="birthDate",

* type="string",

* length = 60,

* nullable=true

* )

*/

protected string | null $birthDate = null;

/**

* @ORM\Column(

* name = "Url",

* type = "string",

* length = 60,

* nullable = true

* )

*/

protected string | null $birthDate = null;
```

```
/**
 * @ORM\Column(
 * name="name",
 * type="string",
 * length = 60,
 * nullable=true
 * )
 */
protected string | null $name = null;
```

Los atributos Url y birthDay los queremos añadir al registrar un usuario pero también se podrán modificar una vez se haya creado, ambos son de tipo string.

```
/**

* @ORM\Column(

* name = "Activo",

* type = "integer",

* length = 60,

* nullable = true

* )

*/
protected int | null $Activo = null;
```

El atributo Activo se utilizará para indicar si el usuario está operativo o si ha sido suspendido, éste es de tipo int y tomará los valores 0, si está "Inactivo" o 1 si está "Activo", siempre que se cree un usuario se creará inactivo, a la espera de que un usuario tipo writer lo active.

Estos atributos se han añadido al constructor:

```
/**

* User constructor.

*

* Oparam string $username username

* Oparam string $email email

* Oparam string $password password

* Oparam string $role Role::ROLE_READER | Role::ROLE_WRITER

* Oparam string|null $name name

* Oparam string|null $birthDate birthDate

* Oparam string|null $Url Url

* Oparam int|null $Activo Activo

*

* Othrows UnexpectedValueException

*/
```

También se han creado los correspondientes métodos setters y getters:

```
/**
  * @return string|null
  */
public function getbirthDate(): ?string
{
    return $this->birthDate;
}

/**
  * @param string|null $birthDate birthDate
  * @return void
  */
public function setbirthDate(?string $birthDate): void
{
    $this->birthDate = $birthDate;
}
```

```
/**
  * @return string|null
  */
public function getUrl(): ?string
{
    return $this->Url;
}

/**
  * @param string|null $Url Url
  * @return void
  */
public function setUrl(?string $Url): void
{
    $this->Url = $Url;
}
```

```
public function getname(): ?string
{
    return $this->name;
}

/**
    * @param string|null $name name
    * @return void
    */
public function setname(?string $name): void
{
    $this->name = $name;
}
```

```
public function getActivo(): ?int
{
    return $this->Activo;
}

/**
    * @param int|null $Activo Activo
    * @return void
    */
public function setActivo(?int $Activo): void
{
    $this->Activo = $Activo;
}
```

Y se han añadido a métodos propios de esta clase que luego serán útiles para crear el código json correspondiente:

En la clase UserController también hemos añadido un método llamado updateUser, que nos permitirá hacer la actualización de la información del usuario en caso de que la cambie,

obviamente el Identificador no se modificará ya que tiene que ser único y es algo interno del sistema además de que permitirlo podría generar inconsistencias con la base de datos.

Además hemos añadido al método put estos nuevos parámetros:

```
if (isset($req_data['birthDate'])) {
    $user->setbirthDate($req_data['birthDate']);
}

if (isset($req_data['Url'])) {
    $user->setUrl($req_data['Url']);
}

if (isset($req_data['Activo'])) {
    $user->setActivo($req_data['Activo']);
}
```

Tanto en ElementBaseController.php como en UserController.php hemos añadido la funcion checkReaderScope:

Esta es llamada desde los métodos get:

```
public function cget(Request $request, Response $response): Response
{
    if (!$this->checkReaderScope($request)) { // 403
        return Error::error($response, statusCode::STATUS_NOT_FOUND);
    }
}
```

Por último a los archivos routersPersons.php, routersEntities.php y routersProducts.php les hemos añadido lo siguiente:

```
->add(JwtMiddleware::class);
```

# Modificación en la base de datos:

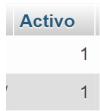
En la tabla user se han añadido 4 columnas:



Para poder dar de alta usuarios se han añadido las columnas "name", "birthDate" y "url", ya que nuestro sistema tendrá que guardar esos datos al registrar un nuevo usuario y asociarlos a su perfil.

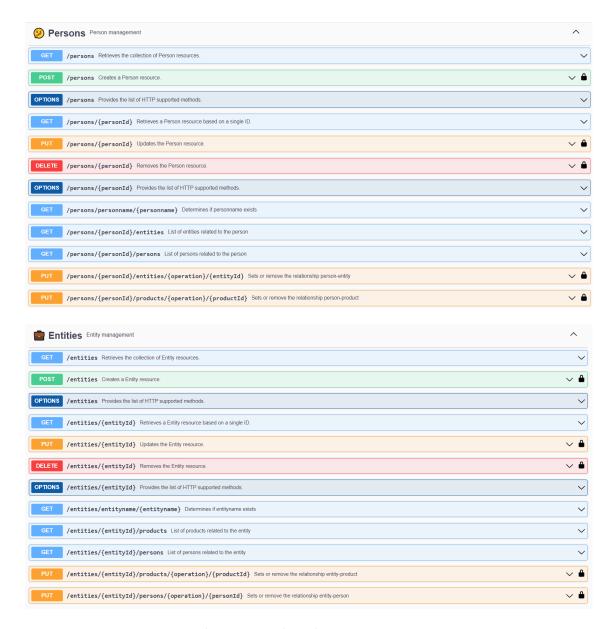
	name	birthDate	Url
1	user	2000-12-25	http://u_image07.example.com/

Adicionalmente se ha creado una columna llamada "Activo" que es la que posteriormente nos proporcionará la funcionalidad de suspender a un usuario temporalmente sin tener que borrar sus datos. Dicha variable sólo será accedida por usuarios tipo writer.



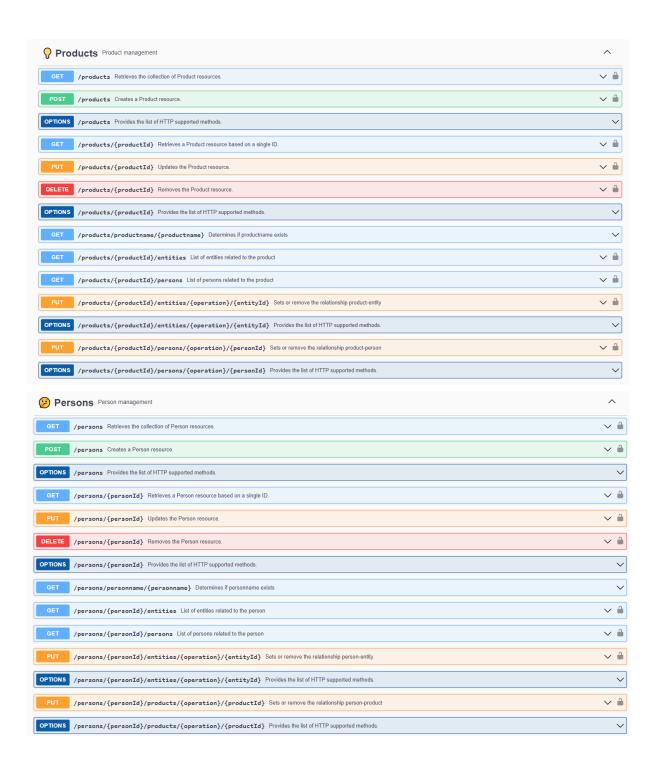
# Cambios realizados en la API:

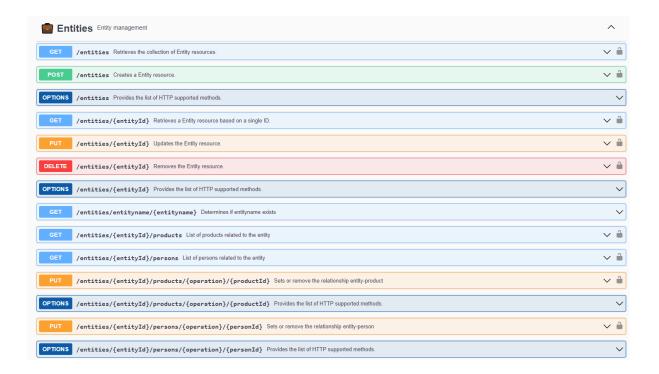
Estos son los esquemas de personas y entidades antes de la modificación:



### Estos son los esquemas después de la modificación:







Como se puede ver después de la modificación se ha añadido seguridad a casi todos los gets; todos los métodos tienen candados lo que quiere decir que son métodos privados a los que solo tienen acceso los usuarios con permisos especiales. Se ha dejado como públicas las operaciones options y el get por nombre de todos los esquemas para poder determinar si existe el nombre de esa entidad, persona...etc. También hemos añadido 2 métodos options en productos, entidades y personas cuya finalidad es solucionar el Problema 3 de la sección problemas encontrados de este documento.

# Aplicación:

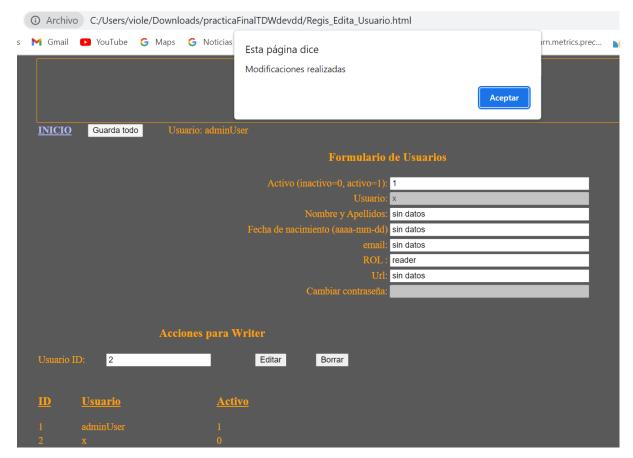
La pantalla de inicio es común a todos los usuarios:



Me registro con un usuario x:

	Anales de la Ciencia
INICIO Guarda todo	
	Formulario de Usuarios
(1) Archivo L Cillipars (male / Downloads (pres	Contraseña (obligatorio): x Usuario (obligatorio): x Nombre y Apellidos: Fecha de nacimiento (aaaa-mm-dd) email: ROL: reader Url:
<ul><li>☑ Archivo   C:/Users/viole/Downloads/prac</li><li>M Gmail</li><li>☑ YouTube</li><li>☑ Maps</li><li>☑ Motion</li></ul>	rias prometric
	Esta página dice  Está usted Inactivado en esta BD. Póngase en contacto con un Writer  Aceptar
Usuario: x Contr	aseña: • Login Registrarse

Se pueden editar los usuarios con permisos de writer, voy a ponerlo en activo:



Al loguearte con un usuario tipo writer te aparece esta pantalla:



Al loguearte con un usuario tipo reader aparecerá esta otra:

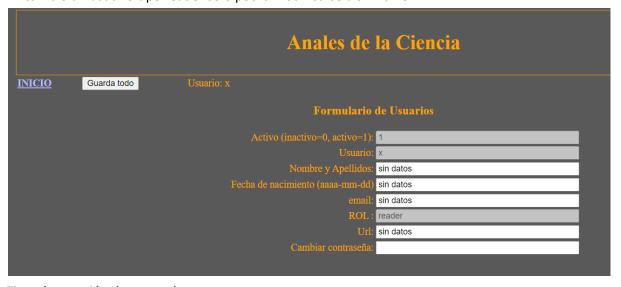


Se puede observar que los usuarios writer pueden añadir, modificar y borrar productos entidades y personas.

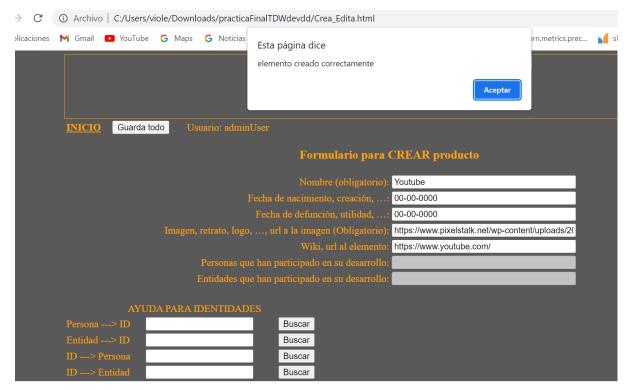
Un usuario writer podrá editar y borrar un usuario por su id:

	Anales de la Ciencia	
<u>INICIO</u> Guarda	a todo Usuario: adminUser	
	Formulario de Usuarios	
	Activo (inactivo=0, activo=1):	
	Usuario:	
	Nombre y Apellidos:	
	Fecha de nacimiento (aaaa-mm-dd)	
	email:	
	ROL:	
	Url:	
	Cambiar contraseña:	
	Acciones para Writer	
Usuario ID:	Editar Borrar	
<u>ID</u> <u>Usuari</u>	Activo	
1 adminUs	er l	

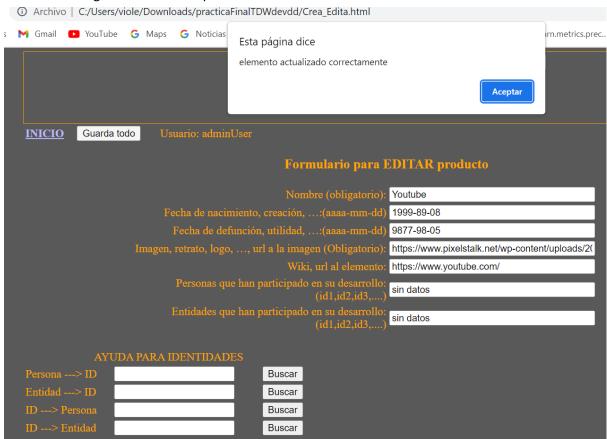
En cambio un usuario tipo reader solo podrá modificarse a sí mismo:



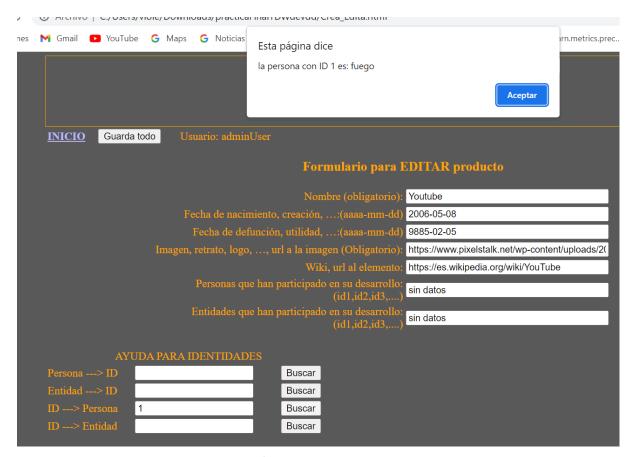
Ejemplo creación de un producto:



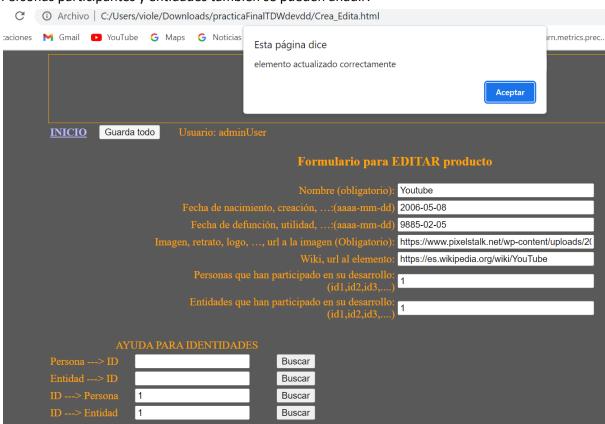
Como no se ha guardado la fecha por tener un formato incorrecto lo edito:



Se puede buscar entidades y personas:



### Personas participantes y entidades también se pueden añadir:



### Resultado:



Un usuario reader lo visualizará de esta forma:



Se puede hacer de forma análoga para personas y entidades.

# Problemas encontrados:

Problema 1: Al modificar el mapeado la tabla user de la base de datos no era modificada, ni tampoco los métodos put, post y delete de la api ya que seguía usando el antiguo mapeado. Lo hemos solucionado borrando la memoria caché de los metadatos utilizando el siguiente comando:

> bin\doctrine orm:clear-cache:metadata

Problema 2: La cabecera de respuesta de la api no permitía ver todas las cabeceras, para solucionarlo hemos añadido una cabecera en la respuesta editando la función process del fichero CorsMiddleware.php:

```
$response = $response->withHeader( name: 'Access-Control-Expose-Headers', value: $requestHeaders ?: '*');
```

Problema 3: El servidor no permitía la conexión por la falta de un preflight al realizar una operación put entre los elementos productos, entidades y personas. Para arreglar esto hemos implementado dos métodos option en estos 3 elementos (productos, entidades y personas), ya que el método options es el encargado del preflight, para lograrlo hemos modificado los siguientes archivos: openapi.yaml, para añadir las 2 funcionalidades

Dentro de la carpeta config:

Routesentities.php: Se han añadido las rutas correspondientes a productos y personas en entidades para que estos 2 métodos options funcionen.

```
// OPTIONS: Provides the list of HTTP supported methods
 $app->options(
     $_ENV['RUTA_API'] . EntityController::PATH_ENTITIES . $REGEX_ENTITY_ID . '/products/rem' . $REGEX_STUFF_ID,
     EntityController::class .
                                ':options'
 )->setName('en1optionsEntity');
     // OPTIONS: Provides the list of HTTP supported methods
 $app->options(
     $_ENV['RUTA_API'] . EntityController::PATH_ENTITIES . $REGEX_ENTITY_ID . '/products/add' . $REGEX_STUFF_ID,
 EntityController::class . '
)->setName('en1optionsEntity');
                                ':options'
     // OPTIONS: Provides the list of HTTP supported methods
 $app->options(
     $_ENV['RUTA_API'] . EntityController::PATH_ENTITIES . $REGEX_ENTITY_ID . '/persons/rem' . $REGEX_STUFF_ID,
     EntityController::class .
                                 ':options'
 )->setName('en2optionsEntity');
    // OPTIONS: Provides the list of HTTP supported methods
$app->options(
    $_ENV['RUTA_API'] . EntityController::PATH_ENTITIES . $REGEX_ENTITY_ID . '/persons/add' . $REGEX_STUFF_ID,
    EntityController::class .
                                ':options'
)->setName('en2optionsEntity');
```

Routesproduct.php: Se han añadido las rutas correspondientes a entidades y personas en productos para que estos 2 métodos options funcionen.

```
// OPTIONS: Provides the list of HTTP supported methods
$app->options(
    $_ENV['RUTA_API'] . ProductController::PATH_PRODUCTS . $REGEX_PRODUCT_ID . '/persons/rem' . $REGEX_STUFF_ID
    ProductController::class . ':options'
)->setName('pr2optionsProduct');
    // OPTIONS: Provides the list of HTTP supported methods
$app->options(
    $_ENV['RUTA_API'] . ProductController::PATH_PRODUCTS . $REGEX_PRODUCT_ID . '/persons/add' . $REGEX_STUFF_ID,
    ProductController::class . ':options'
)->setName('pr2optionsProduct');
    // OPTIONS: Provides the list of HTTP supported methods
$app->options(
    $_ENV['RUTA_API'] . ProductController::PATH_PRODUCTS . $REGEX_PRODUCT_ID . '/entities/rem' . $REGEX_STUFF_ID,
    ProductController::class . ':options'
)->setName('pr1optionsProduct');
    // OPTIONS: Provides the list of HTTP supported methods
$app->options(
    $_ENV['RUTA_API'] . ProductController::PATH_PRODUCTS . $REGEX_PRODUCT_ID . '/entities/add' . $REGEX_STUFF_ID,
    ProductController::class . ':options'
)->setName('pr1optionsProduct');
```

Routespersons.php: Se han añadido las rutas correspondientes a productos y entidades en personas para que estos 2 métodos options funcionen.

```
// OPTIONS: Provides the list of HTTP supported methods
    $app->options(
    $_ENV['RUTA_API'] . PersonController::PATH_PERSONS . $REGEX_PERSON_ID . '/products/rem' . $REGEX_STUFF_ID,
    PersonController::class . ':options'
)->setName('pe2optionsPerson');
    // OPTIONS: Provides the list of HTTP supported methods
    $app->options(
    $_ENV['RUTA_API'] . PersonController::PATH_PERSONS . $REGEX_PERSON_ID . '/products/add' . $REGEX_STUFF_ID,
    PersonController::class . ':options'
)->setName('pe2optionsPerson');
    // OPTIONS: Provides the list of HTTP supported methods
$app->options(
    $_ENV['RUTA_API'] . PersonController::PATH_PERSONS . $REGEX_PERSON_ID . '/entities/rem' . $REGEX_STUFF_ID,
    PersonController::class . ':options'
)->setName('pe2optionsPerson');
   // OPTIONS: Provides the list of HTTP supported methods
$app->options(
   $_ENV['RUTA_API'] . PersonController::PATH_PERSONS . $REGEX_PERSON_ID . '/entities/add' . $REGEX_STUFF_ID,
   PersonController::class . ':options'
)->setName('peloptionsPerson');
// PUT /persons/{personId}/entities/add/{stuffId}
```

Src\controller\elementrelationsbasecontroller.php: Hemos añadido la función options: