

# MEMORIA DEL TRABAJO DE PHP DE TDW

REALIZADO POR VIOLETA MACIAS DE MIGUEL Y DAVID RAMAJO FERNANDEZ

## Introducción

Este trabajo consiste en una api que da servicios externos para que se pueda conectar alguien y que pueda hacer cambios en la base de datos sin tener que conocer el funcionamiento interno de esta. Este proyecto utiliza un componente Dotenv para realizar la configuración a través de variables de entorno.

Hay que generar un esquema de base de datos vacío y un usuario/contraseña con privilegios completos sobre dicho esquema, realizar una copia del fichero `./env`, renombrarla como `./envn.local` y editando este fichero, modificar las variables `DATABASE_NAME`, `DATABASE_USER` Y `DATABASE_PASSWD`, con los valores correspondientes. A continuación abrimos en el terminal del ordenador el directorio raíz del proyecto y ejecutamos los comandos:

```
> 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.

```
> bin/doctrine orm:schema:update --dump-sql --force
```

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:validate
```

Este comando se utiliza para verificar la validez de la información de mapeo y la sincronización con la base de datos.

Para la ejecución de las pruebas;

Hemos visto que la aplicación incorpora un conjunto completo de herramientas para la ejecución de pruebas unitarias y de integración con PHPUnit. Después, empleando este conjunto de herramientas, hemos podido comprobar de manera automática, el correcto funcionamiento de la API y sin necesidad de herramientas adicionales.

Para continuar y poder configurar el entorno de pruebas, hemos creado otro esquema de bases de datos vacío, y una copia del fichero `./phpunit.xml.dist` y renombrarla como `./phpunit.xml`. Además hemos editado este fichero para asignar los siguientes parámetros:

Configuración del acceso a la nueva base de datos añadiendo los datos correspondientes a las variables `DATABASE_NAME`, `DATABASE_USER` y `DATABASE_PASSWD`.

Hemos modificado el nombre y contraseña de los usuarios que se utilizan en la realización de las pruebas.

Para poder lanzar suite de pruebas se ejecuta el comando:

```
> bin/phpunit [--testdox] [--coverage-test] [v]
```

Adicionalmente para poder comprobar la calidad de las pruebas, el proyecto incluye test de mutaciones, generados con la herramienta de infection. El funcionamiento es simple: se generan pequeños cambios en el código original (mutantes), y a continuación se ejecuta la

batería de pruebas. Por consiguiente, las pruebas fallan, indica que ha sido capaz de detectar la modificación del código, y el mutante es eliminado. Sin embargo, si pasa las pruebas, el mutante sobrevive y la fiabilidad de la prueba queda totalmente cuestionada.

Para lanzar los test de mutaciones se ejecuta el comando:

```
>composer infection
```

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 <http://127.0.0.1:8000/> (direccion ip y puerto 8000).

## Explicación del código:

Para comenzar, todas las funciones que hemos implementado reciben un objeto request, un objeto response y un array llamado “args” que devuelven, un objeto response que es la respuesta que se le devolverá al usuario y que contendrá la información correspondiente contestando a los datos que ha introducido el usuario (request).

Como en todas estas funciones devuelven el objeto response que contiene y hace uso de las funciones error y withJson vamos a explicar éstas a continuación.

El objeto response tiene un número de error (\$response) y un string con los caracteres de la respuesta http (\$streamFactory).

```
public function getBody()
{
    return $this->response->getBody();
}

/**
 * {@inheritdoc}
 */
public function getHeader($name): array
{
    return $this->response->getHeader($name);
}
```

También tiene un **header** con toda la información correspondiente a la **request** y un **body** en el que se guarda un json con la respuesta que le devolvemos al usuario.

```

public static function error(
    Response $response,
    int $statusCode
): Response {
    return $response
        ->withJson(
            [
                'code' => $statusCode,
                'message' => self::MESSAGES[$statusCode]
            ],
            $statusCode
        );
}

```

Añade el **código** con su mensaje de **error** correspondiente en un json en caso de que falle.

```

public function withJson($data, ?int $status = null, int $options = 0, int $depth = 512): ResponseInterface
{
    $json = (string) json_encode($data, $options, $depth);

    if (json_last_error() !== JSON_ERROR_NONE) {
        throw new RuntimeException(json_last_error_msg(), json_last_error());
    }

    $response = $this->response
        ->withHeader( name: 'Content-Type', value: 'application/json')
        ->withBody($this->streamFactory->createStream($json));

    if ($status !== null) {
        $response = $response->withStatus($status);
    }

    return new static($response, $this->streamFactory);
}

```

Añade el **código** con la lista de objetos correspondiente en un json en caso de que todo funcione correctamente.

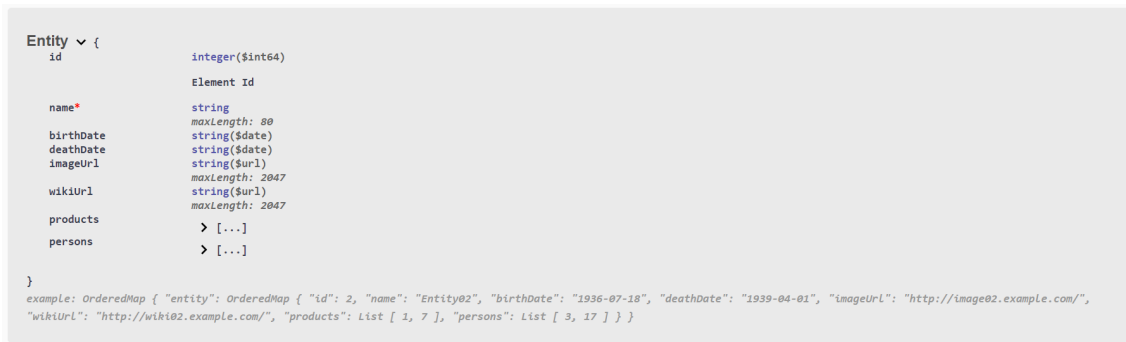
En los **esquemas Person y entity**, hemos implementado los métodos **get**, **post**, **options**, **put** y **delete**

```

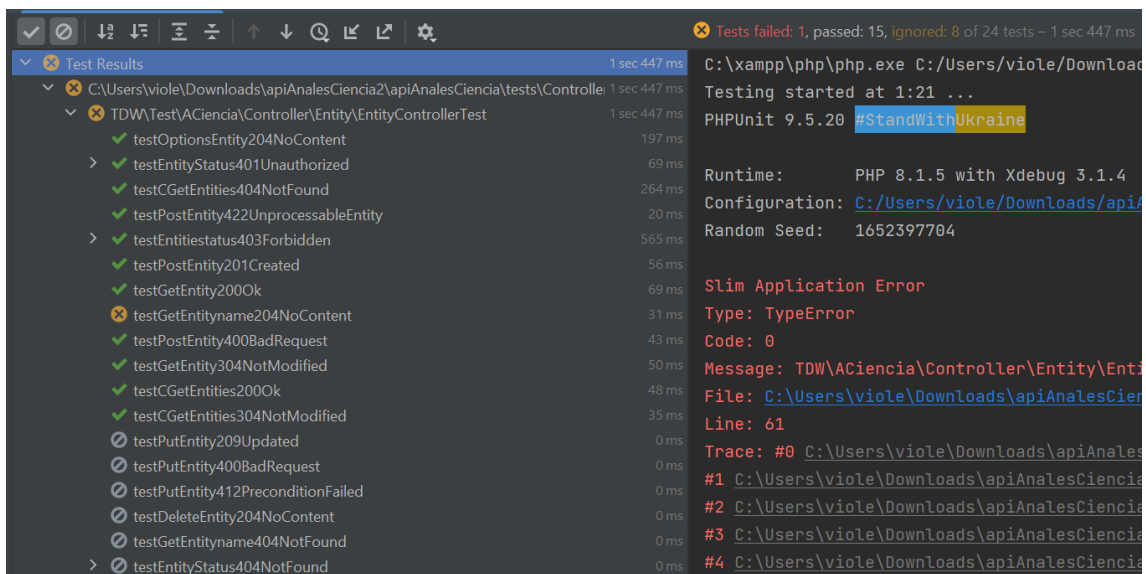
Person {
  id integer($int64)
  Element Id
  name* string
    maxLength: 80
  birthDate string($date)
  deathDate string($date)
  imageUrl string($url)
    maxLength: 2047
  wikiUrl string($url)
    maxLength: 2047
  products > [...]
  entities > [...]
}

example: OrderedMap { "person": OrderedMap { "id": 17, "name": "Person17", "birthDate": "2017-07-17", "deathDate": "2701-01-07", "imageUrl": "http://image17.example.com/", "wikiUrl": "http://wiki17.example.com/", "products": List [ 1, 7 ], "entities": List [ 2 ] } }

```



Los test quedan así:



En la práctica nos encontramos las funciones `getPersonname` y `getEntityname` de las clases `PersonController` y `EntityController` respectivamente que faltan por implementar:

```
public function getPersonname(Request $request, Response $response, array $args): Response
{
    // @TODO
}
```

```
public function getEntityname(Request $request, Response $response, array $args): Response
{
    // @TODO
}
```

Al completarlas quedan así:

```
public function getPersonname(Request $request, Response $response, array $args): Response
{
    return $this->getElementByName($response, $args['personname']);
}
```

```
public function getEntityname(Request $request, Response $response, array $args): Response
{
    return $this->getElementByName($response, $args['entityname']);
}
```

Tanto la clase EntityController como la clase PersonController heredan de la clase ElementBaseController.

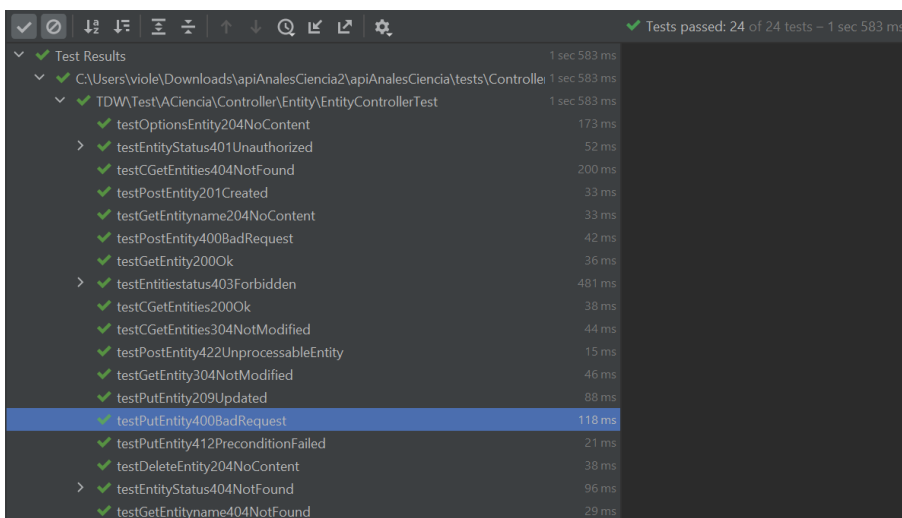
En las funciones getPersonname y getEntityname podemos ver que devuelve un objeto response con un código de 3 dígitos (ejemplo 204) que indica tanto en una función como en la otra si hay un error o ha tenido éxito y devolverá un json con la lista de todas las personas en el getPersonname o con la lista de todas las entidades en el getEntityname que tienen el nombre que el usuario solicitó en el request.

En caso de no funcionar se devolverá un json con el código de error y el mensaje correspondiente a ese error que se ha producido.

La función getProductname de la clase ProductController funciona igual solo que devolverá la lista de productos en caso de encontrar el nombre que solicitó el usuario:

```
public function getProductname(Request $request, Response $response, array $args): Response
{
    return $this->getElementByName($response, $args['productname']);
}
```

Cuando paso los test quedan así:



Test Name	Duration
Test Results	1 sec 583 ms
C:\Users\viola\Downloads\apiAnalesCiencia2\apiAnalesCiencia\tests\Controller	1 sec 583 ms
TDW\Test\ACiencia\Controller\Entity\EntityControllerTest	1 sec 583 ms
testOptionsEntity204NoContent	173 ms
testEntityStatus401Unauthorized	52 ms
testGetEntities404NotFound	200 ms
testPostEntity201Created	33 ms
testGetEntityname204NoContent	33 ms
testPostEntity400BadRequest	42 ms
testGetEntity200Ok	36 ms
testEntityStatus403Forbidden	481 ms
testGetEntities200Ok	38 ms
testGetEntities304NotModified	44 ms
testPostEntity422UnprocessableEntity	15 ms
testGetEntity304NotModified	46 ms
testPutEntity209Updated	88 ms
testPutEntity400BadRequest	118 ms
testPutEntity412PreconditionFailed	21 ms
testDeleteEntity204NoContent	38 ms
testEntityStatus404NotFound	96 ms
testGetEntityname404NotFound	29 ms

También hemos modificado las funciones de las clases EntityRelationsController y PersonRelationsController que heredan de la clase ElementRelationsBaseController que se muestran a continuación:

De la clase EntityRelationsController hemos implementado las funciones:

```

public function getPersons(Request $request, Response $response, array $args): Response
{
    // @TODO
}

public function operationPerson(Request $request, Response $response, array $args): Response
{
    // @TODO
}

public function getProducts(Request $request, Response $response, array $args): Response
{
    // @TODO
}

public function operationProduct(Request $request, Response $response, array $args): Response
{
    // @TODO
}

```

getPersons(Request \$request, Response \$response, array \$args): Response

Devuelve una response con un archivo json que contiene todas las personas (body).

```

public function getPersons(Request $request, Response $response, array $args): Response
{
    $elementData = [
        'getter' => 'getPersons',
        'stuff' => PersonController::getEntitiesTag(),
    ];
    return $this->getElements($response, $args, $elementData); // @TODO
}

```

operationPerson(Request \$request, Response \$response, array \$args): Response

Devuelve una response con un head que incluye la información de las operaciones que se pueden hacer con una persona.

```

public function operationPerson(Request $request, Response $response, array $args): Response
{
    $elementData = [
        'stuffENAME' => PersonController::getEntityClassName(),
        'stuffId' => $args['stuffId'],
        'getter' => 'getPersons',
        'stuff' => PersonController::getEntitiesTag(),
    ];
    return $this->operationStuff($request, $response, $args, $elementData); // @TODO
}

```

getProducts(Request \$request, Response \$response, array \$args): Response

Devuelve una response con un archivo json que contiene todos los productos (body).

```

public function getProducts(Request $request, Response $response, array $args): Response
{
    $elementData = [
        'getter' => 'getProducts',
        'stuff' => ProductController::getEntitiesTag(),
    ];
    return $this->getElements($response, $args, $elementData);
}

```

operationProduct(Request \$request, Response \$response, array \$args): Response

Devuelve una response que incluye la información de las operaciones que se pueden hacer con un producto (head).

```

public function operationProduct(Request $request, Response $response, array $args): Response
{
    $elementData = [
        'stuffEName' => ProductController::getEntityClassName(),
        'stuffId' => $args['stuffId'],
        'getter' => 'getProducts',
        'stuff' => ProductController::getEntitiesTag(),
    ];
    return $this->operationStuff($request, $response, $args, $elementData); // @TODO
}

```

De la clase PersonRelationsController:

```

public function getEntities(Request $request, Response $response, array $args): Response
{
    // @TODO
}

public function operationEntity(Request $request, Response $response, array $args): Response
{
    // @TODO
}

public function getProducts(Request $request, Response $response, array $args): Response
{
    // @TODO
}

public function operationProduct(Request $request, Response $response, array $args): Response
{
    // @TODO
}

```

getEntities(Request \$request, Response \$response, array \$args): Response

Devuelve una response con un archivo json que contiene todas las personas (body).



```

public function getEntities(Request $request, Response $response, array $args): Response
{
    $elementData = [
        'getter' => 'getEntities',
        'stuff' => EntityController::getEntitiesTag(),
    ];
    return $this->getElements($response, $args, $elementData);
}

```

operationEntity(Request \$request, Response \$response, array \$args): Response

Devuelve una response con un head que incluye la información de las operaciones que se pueden hacer con una entidad

```

public function operationEntity(Request $request, Response $response, array $args): Response
{
    $elementData = [
        'stuffENAME' => EntityController::getEntityClassName(),
        'stuffId' => $args['stuffId'],
        'getter' => 'getEntities',
        'stuff' => EntityController::getEntitiesTag(),
    ];
    return $this->operationStuff($request, $response, $args, $elementData); // @TODO
}

```

getProducts(Request \$request, Response \$response, array \$args): Response

Devuelve una response con un archivo json que contiene todos los productos (body)

```

public function getProducts(Request $request, Response $response, array $args): Response
{
    $elementData = [
        'getter' => 'getProducts',
        'stuff' => ProductController::getEntitiesTag(),
    ];
    return $this->getElements($response, $args, $elementData);
}

```

operationProduct(Request \$request, Response \$response, array \$args): Response

Devuelve una response con un head que incluye la información de las operaciones que se pueden hacer con un producto

```

public function operationProduct(Request $request, Response $response, array $args): Response
{
    $elementData = [
        'stuffENAME' => ProductController::getEntityClassName(),
        'stuffId' => $args['stuffId'],
        'getter' => 'getProducts',
        'stuff' => ProductController::getEntitiesTag(),
    ];
    return $this->operationStuff($request, $response, $args, $elementData);
}

```

Todas estas funciones modificadas llaman a otras funciones, que a su vez pueden llamar a otras funciones y casi todas devuelven un objeto response que van modificando en cada una de las llamadas. En este objeto response se almacena tanto el número de error o éxito y un código json correspondiente que tendrá la información del error o en caso de haber tenido éxito con los datos que se quieren devolver al usuario.

## Explicación del funcionamiento de la aplicación:

En el esquema personas, ofrece estas operaciones al usuario:

	<b>Persons</b>	Person management	^
GET	/persons	Retrieves the collection of Person resources.	▼
POST	/persons	Creates a Person resource.	▼ 🔒
OPTIONS	/persons	Provides the list of HTTP supported methods.	▼
GET	/persons/{personId}	Retrieves a Person resource based on a single ID.	▼
PUT	/persons/{personId}	Updates the Person resource.	▼ 🔒
DELETE	/persons/{personId}	Removes the Person resource.	▼ 🔒
OPTIONS	/persons/{personId}	Provides the list of HTTP supported methods.	▼
GET	/persons/personname/{personname}	Determines if personname exists	▼
GET	/persons/{personId}/entities	List of entities related to the person	▼
GET	/persons/{personId}/persons	List of persons related to the person	▼
PUT	/persons/{personId}/entities/{operation}/{entityId}	Sets or remove the relationship person-entity	▼ 🔒
PUT	/persons/{personId}/products/{operation}/{productId}	Sets or remove the relationship person-product	▼ 🔒

En el **get/persons** devuelve al usuario, una lista con todas las personas que hay creadas en un json:

```
200
Response body
{
  "persons": [
    {
      "person": {
        "id": 1,
        "name": "ana",
        "birthdate": "1998-09-04",
        "deathdate": "2022-05-13",
        "imageUrl": "https://upload.wikimedia.org/wikipedia/commons/thumb/f/fe/Anne_Frank_lacht_naar_de_schoolfotograaf.jpg/375px-Anne_Frank_lacht_naar_de_schoolfotograaf.jpg",
        "wikiUrl": "https://es.wikipedia.org/wiki/Ana_Frank",
        "products": null,
        "entities": null
      }
    },
    {
      "person": {
        "id": 2,
        "name": "Fibonacci",
        "birthdate": null,
        "deathdate": null,
        "imageUrl": "https://upload.wikimedia.org/wikipedia/commons/thumb/8/8e/Leonardo_da_Pisa.jpg/330px-Leonardo_da_Pisa.jpg",
        "wikiUrl": "https://en.wikipedia.org/wiki/Fibonacci#Biography",
        "products": null,
        "entities": {
          "": 1
        }
      }
    }
  ]
}

Response headers
access-control-allow-credentials: true
access-control-allow-headers: *
access-control-allow-methods: GET,POST,OPTIONS
access-control-allow-origin: *
cache-control: private
connection: close
content-type: application/json
date: Fri, 13 May 2022 22:35:22 GMT
etag: e5c2a505e8a0d70019eba3c2ad0d5
host: 127.0.0.1:8000
x-powered-by: PHP/8.1.5

Request duration
1317 ms
```

En el **post/persons**, se creará una persona introduciendo las datos correspondientes a esta en formato json:

Para crear el usuario Ana Frank:

**POST** /persons Creates a Person resource.

Creates a new person

Parameters Cancel Reset

No parameters

Request body application/json

Element data

```
{
  "name": "ana",
  "birthDate": "1990-09-04",
  "deathDate": "2022-05-13",
  "imageUrl": "https://upload.wikimedia.org/wikipedia/commons/thumb/f/fe/Anne_Frank_lacht_naar_de_schoolfotograaf.jpg/375px-Anne_Frank_lacht_naar_de_schoolfotograaf.jpg",
  "wikiUrl": "https://es.wikipedia.org/wiki/Ana_Frank"
}
```

```
{
  "name": "ana",
  "birthDate": "1990-09-04",
  "deathDate": "2022-05-13",
  "imageUrl":
    "https://upload.wikimedia.org/wikipedia/commons/thumb/f/fe/Anne_Frank_lacht_naar_de_schoolfotograaf.jpg/375px-Anne_Frank_lacht_naar_de_schoolfotograaf.jpg",
  "wikiUrl": "https://es.wikipedia.org/wiki/Ana_Frank"
}
```

Server response

Code	Details
201	<p>Response body</p> <pre>{   "person": {     "id": 1,     "name": "ana",     "birthDate": "1990-09-04",     "deathDate": "2022-05-13",     "imageUrl": "https://upload.wikimedia.org/wikipedia/commons/thumb/f/fe/Anne_Frank_lacht_naar_de_schoolfotograaf.jpg/375px-Anne_Frank_lacht_naar_de_schoolfotograaf.jpg",     "wikiUrl": "https://es.wikipedia.org/wiki/Ana_Frank",     "products": null,     "entities": null   } }</pre> <p>Response headers</p> <pre>access-control-allow-credentials: true access-control-allow-headers: * access-control-allow-methods: GET,POST,OPTIONS access-control-allow-origin: * connection: close content-type: application/json date: Fri, 13 May 2022 16:30:34 GMT host: 127.0.0.1:8000 location: http://127.0.0.1:8000/api/v1/persons/1 x-powered-by: PHP/8.1.5</pre> <p>Request duration</p> <p>349 ms</p>

Creación usuario Fibonacci

Server response

Code	Details
201	<p>Response body</p> <pre>{   "person": {     "id": 2,     "name": "Fibonacci",     "birthDate": null,     "deathDate": null,     "imageUrl": "https://upload.wikimedia.org/wikipedia/commons/thumb/8/8e/Leonardo_da_Pisa.jpg/330px-Leonardo_da_Pisa.jpg",     "wikiUrl": "https://en.wikipedia.org/wiki/Fibonacci#Biography",     "products": null,     "entities": null   } }</pre> <p>Response headers</p> <pre>access-control-allow-credentials: true access-control-allow-headers: * access-control-allow-methods: GET,POST,OPTIONS access-control-allow-origin: * connection: close content-type: application/json date: Fri,13 May 2022 16:43:53 GMT host: 127.0.0.1:8000 location: http://127.0.0.1:8000/api/v1/persons/2 x-powered-by: PHP/8.1.5</pre> <p>Request duration</p> <p>344 ms</p>

```
{
  "name": "Fibonacci",
  "birthDate": "1170",
  "deathDate": "1250",
  "imageUrl":
"https://upload.wikimedia.org/wikipedia/commons/thumb/8/8e/Leonardo_da_Pisa.jpg/330px-Leonardo_da_Pisa.jpg",
  "wikiUrl": "https://en.wikipedia.org/wiki/Fibonacci#Biography"
}
```

Server response

Code	Details
201	<p>Response body</p> <pre>{   "person": {     "id": 2,     "name": "Fibonacci",     "birthDate": null,     "deathDate": null,     "imageUrl": "https://upload.wikimedia.org/wikipedia/commons/thumb/8/8e/Leonardo_da_Pisa.jpg/330px-Leonardo_da_Pisa.jpg",     "wikiUrl": "https://en.wikipedia.org/wiki/Fibonacci#Biography",     "products": null,     "entities": null   } }</pre> <p>Response headers</p> <pre>access-control-allow-credentials: true access-control-allow-headers: * access-control-allow-methods: GET,POST,OPTIONS access-control-allow-origin: * connection: close content-type: application/json date: Fri,13 May 2022 16:43:53 GMT host: 127.0.0.1:8000 location: http://127.0.0.1:8000/api/v1/persons/2 x-powered-by: PHP/8.1.5</pre> <p>Request duration</p> <p>344 ms</p>

Error al crear una **persona** igual:

Server response

Code	Details
400	<p>Error: Bad Request</p> <p>Response body</p> <pre>{   "code": 400,   "message": "BAD REQUEST: name or e-mail already exists, or role does not exist" }</pre> <p>Response headers</p> <pre>access-control-allow-credentials: true access-control-allow-headers: * access-control-allow-methods: GET,POST,OPTIONS access-control-allow-origin: * connection: close content-type: application/json date: Fri, 13 May 2022 16:49:19 GMT host: 127.0.0.1:8000 x-powered-by: PHP/8.1.5</pre>

Opciones de personas: en **options/persons**, se mostrará las opciones que puede hacer el usuario con los objetos persona.

**OPTIONS** /persons Provides the list of HTTP supported methods. ^

Return a **Allow** header with a comma separated list of HTTP supported methods.

**Parameters** Cancel

No parameters

Execute Clear

## Server response

### Code

### Details

204

### Response headers

```
access-control-allow-credentials: true
access-control-allow-headers: *
access-control-allow-methods: GET,POST,OPTIONS
access-control-allow-origin: *
allow: GET,POST,OPTIONS
cache-control: private
connection: close
content-type: text/html; charset=UTF-8
date: Fri,13 May 2022 16:46:28 GMT
host: 127.0.0.1:8000
x-powered-by: PHP/8.1.5
```

### Request duration

244 ms

## Responses

En el `get/persons/{personid}`, le pasas un id y te devuelve a la persona correspondiente.

Code	Details
200	<div><div>Response body</div><pre>{   "person": {     "id": 1,     "name": "ana",     "birthDate": "1990-09-04",     "deathDate": "2022-05-13",     "imageUrl": "https://upload.wikimedia.org/wikipedia/commons/thumb/f/fe/Anne_Frank_lacht_naar_de_schoolfotograaf.jpg/375px-Anne_Frank_lacht_naar_de_schoolfotograaf.jpg",     "wikiUrl": "https://es.wikipedia.org/wiki/Ana_Frank",     "products": null,     "entities": null   } }</pre></div> <div><div>Response headers</div><pre>access-control-allow-credentials: true access-control-allow-headers: * access-control-allow-methods: GET,DELETE,OPTIONS,PUT access-control-allow-origin: * cache-control: private connection: close content-type: application/json date: Fri,13 May 2022 22:44:16 GMT etag: d1061c11aee12d6a907b82af6141fe6 host: 127.0.0.1:8000 x-powered-by: PHP/8.1.5</pre></div> <div><div>Request duration</div><p>975 ms</p></div>

En el `put/persons/{personid}`, le pasas un id de la persona que quieres actualizar.

En el `delete/persons/{personid}`, le pasas un id de la persona que quieres eliminar:

Request URL

```
http://127.0.0.1:8000/api/v1/persons/1
```

Server response

Code	Details
204	<p>Response headers</p> <pre>access-control-allow-credentials: true access-control-allow-headers: * access-control-allow-methods: GET,DELETE,OPTIONS,PUT access-control-allow-origin: * connection: close content-type: text/html; charset=UTF-8 date: Fri,13 May 2022 22:54:34 GMT host: 127.0.0.1:8000 x-powered-by: PHP/8.1.5</pre> <p>Request duration</p> <pre>1138 ms</pre>

En **options/persons/{personid}**, se mostrará las opciones con el objeto persona el cual el usuario le ha pasado el id.

En **persons/personsname/{personsname}**, se busca una persona por nombre para comprobar si existe.

En **/persons/{personid}/entities**, devuelve una lista de entidades relacionadas con una persona, la cual su id ha introducido el usuario.

En **/persons/{personid}/persons**, el usuario introduce un id de una persona y este método devuelve una lista de personas relacionadas con la persona la cual el usuario ha querido buscar.

En **/persons/{personid}/entities/{operation}/{entityid}**, introduce o elimina una relación entre persona y entidad.

PUT

/persons/{personId}/entities/{operation}/{entityId} Sets or remove the relationship person-entity

⌵ 🔒

Establishes/Removes the relationship of the person with the entity

Parameters

Cancel

Name	Description
<b>personId</b> <small>required</small> integer(\$int64) (path)	ID of person
<input type="text" value="2"/>	
<b>operation</b> <small>required</small> (path)	Set the operation to perform:
<div><div>• add: add the relationship</div><div>• rem: remove the relationship</div></div> <div><input type="text" value="add"/></div>	
<b>entityId</b> <small>required</small> integer(\$int64) (path)	ID of entity
<input type="text" value="1"/>	

Execute

Clear

Request URL

http://127.0.0.1:8000/api/v1/persons/2/entities/add/1

Server response

Code	Details
209	<div>Response body</div> <pre>{   "person": {     "id": 2,     "name": "Fibonacci",     "birthDate": null,     "deathDate": null,     "imageUrl": "https://upload.wikimedia.org/wikipedia/commons/thumb/8/8e/Leonardo_da_Pisa.jpg/330px-Leonardo_da_Pisa.jpg",     "wikiUrl": "https://en.wikipedia.org/wiki/Fibonacci#Biography",     "products": null,     "entities": [       1     ]   } }</pre> <div>Response headers</div> <pre>access-control-allow-credentials: true access-control-allow-headers: * access-control-allow-methods: PUT access-control-allow-origin: * connection: close content-type: application/json date: Fri, 13 May 2022 22:28:02 GMT host: 127.0.0.1:8000 x-powered-by: PHP/8.1.5</pre> <div>Request duration</div> <div>1185 ms</div>

En `/persons/{personId}/products/{operation}/{productid}`, introduce o elimina una relación entre persona y producto.



**Para entidades,** lo mismo que hemos explicado en otros conceptos, con la única diferencia que en vez de para personas es para entidades.

Entities Entity management			^
GET	/entities	Retrieves the collection of Entity resources.	▼
POST	/entities	Creates a Entity resource.	▼ 🔒
OPTIONS	/entities	Provides the list of HTTP supported methods.	▼
GET	/entities/{entityId}	Retrieves a Entity resource based on a single ID.	▼
PUT	/entities/{entityId}	Updates the Entity resource.	▼ 🔒
DELETE	/entities/{entityId}	Removes the Entity resource.	▼ 🔒
OPTIONS	/entities/{entityId}	Provides the list of HTTP supported methods.	▼
GET	/entities/entityname/{entityname}	Determines if entityname exists	▼
GET	/entities/{entityId}/products	List of products related to the entity	▼
GET	/entities/{entityId}/persons	List of persons related to the entity	▼
PUT	/entities/{entityId}/products/{operation}/{productId}	Sets or remove the relationship entity-product	▼ 🔒
PUT	/entities/{entityId}/persons/{operation}/{personId}	Sets or remove the relationship entity-person	▼ 🔒

Todos los métodos que tienen candados, son métodos privados a los que solo tienen acceso los usuarios con permisos especiales, es decir, el usuario administrador, lo cual el resto podrán ser accedidos por los usuarios.

Todas estas opciones que utiliza el usuario, independientemente si es el administrador, envía un **request URL (petición)** que lo que hace es que añade la información que el usuario ha introducido en la **dirección URL** para realizar dicha petición, y devolverá un response body, un response headers; el response body tiene un json y el response headers la información correspondiente a la request.

**El response body y headers.** está explicado al principio de la explicación del código, incluidos en el objeto response (funciones de este objeto).

#### Request URL

```
http://127.0.0.1:8000/api/v1/persons/2/entities/add/1
```