

Memoria – Práctica Back-End API REST

Nombre del estudiante: David Kai Ming Lew Pérez

Memoria – Práctica Back-End API REST

Pasos Iniciales

He configurado todo como mencionaba el README del proyecto y procedo a instalar todas las dependencias de composer y crear el schema con Doctrine y phpMyAdmin.

Completando Entities y Persons

Inicialmente, comienzo implementando los TO DO que hay en las clases EntityRelationsController y PersonRelationsController.

Implementacion GET y PUT Entity

Seguindo la lógica proporcionada en el código base con Products, he aplicado la misma lógica para completarlos:

GET /entities/{entityId}/persons

Es el metodo que se ocupa de devolver las personas relacionadas con una entidad en concreto.

Primero recibe una petición en la que se pide ver las personas asociadas a la entidad, seguido de la comprobación de que el ID sea válido, si no devuelve una lista vacía. Se verifica a continuación que encuentre una Entidad con el id proporcionado. Finalmente, devuelve la lista de personas (o vacía si no había ninguna) como respuesta al que hizo la petición.

PUT /entities/{entityId}/persons/add/{elementId}

PUT /entities/{entityId}/persons/rem/{elementId}

Inicialmente recibe una petición para añadir o eliminar una persona de una entidad. Luego extrae y valida los identificadores y el tipo de operación. Si todo es correcto, realiza la acción sobre la relación y devuelve la entidad actualizada como respuesta.

GET /entities/{entityId}/products

Primero recibe una petición en la que se pide ver los productos asociados a una entidad. Luego extrae y valida el identificador de la entidad. Si el ID no es válido, devuelve una lista vacía. A continuación, busca la entidad correspondiente y obtiene su lista de productos. Finalmente, devuelve esa lista (o vacía si no hay productos) como respuesta.

PUT /entities/{entityId}/products/add/{elementId}

PUT /entities/{entityId}/products/rem/{elementId}

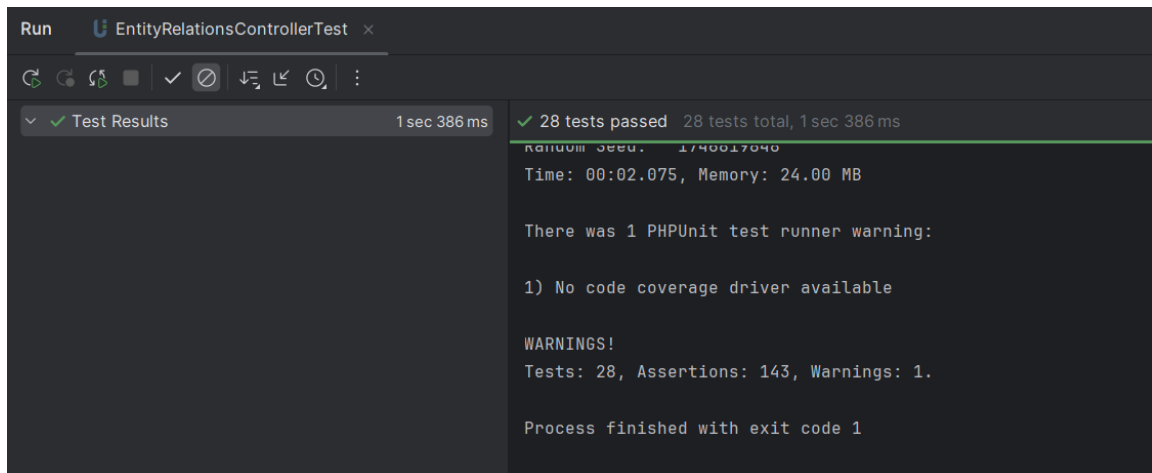
Primero recibe una petición para añadir o eliminar un producto asociado a una entidad. Luego extrae y valida los identificadores y la operación solicitada. Si todo es correcto, realiza la acción sobre la relación entre la entidad y el producto, y devuelve la entidad actualizada como respuesta.

Implementacion GET y PUT Persons

La logica de la implementación es la misma que Entity, solo que aquí es tener en cuenta al revés la relación

Tests realizados

Comprobando los resultados de los test se comprueba que cumplen con la funcionalidad



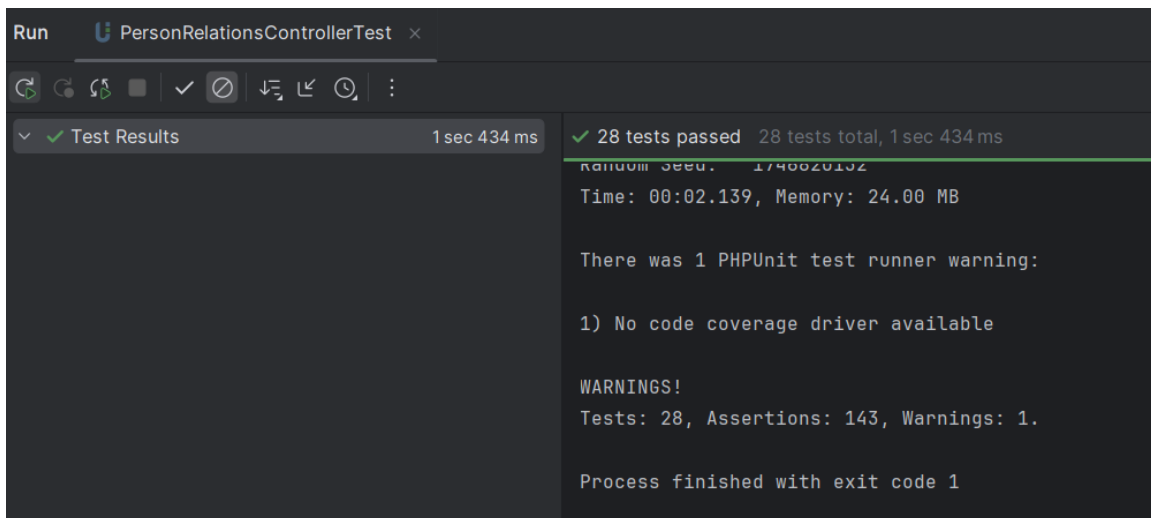
```
Run EntityRelationsControllerTest x
Test Results 1 sec 386 ms
✓ 28 tests passed 28 tests total, 1 sec 386 ms
Random Seed: 1740017040
Time: 00:02.075, Memory: 24.00 MB

There was 1 PHPUnit test runner warning:

1) No code coverage driver available

WARNINGS!
Tests: 28, Assertions: 143, Warnings: 1.

Process finished with exit code 1
```



```
Run PersonRelationsControllerTest x
Test Results 1 sec 434 ms
28 tests passed 28 tests total, 1 sec 434 ms
Random Seed: 1740020132
Time: 00:02.139, Memory: 24.00 MB

There was 1 PHPUnit test runner warning:

1) No code coverage driver available

WARNINGS!
Tests: 28, Assertions: 143, Warnings: 1.

Process finished with exit code 1
```

Asociaciones

El procedimiento ha sido primero la creación de la Entity/Asociacion.

He seguido la estructura de las demás clases de separar las responsabilidades en 3 clases distintas, siendo AsociacionCommandController, AsociacionQueryController, AsociacionRelationsController.

El command se ocupa de crear, modificar y eliminar las asociaciones (métodos POST, PUT, DELETE, OPTIONS)

El query se ocupa en consultar los datos de las asociaciones (métodos cget, get, getByNombre y options).

El relations se ocupa de gestionar las relaciones entre una asociación y otras entidades (métodos getEntities, putEntity, deleteEntity, operationEntity y options)

También defino un AsociacionFactory en la carpeta Factory que se encargue de la lógica de la creación.

Aquí es donde más problemas he tenido, ya que era necesario definir un nombre y una url de su sitio web. En las demás clases que había en la carpeta Entity, todas se extendían de Element, pero justo para el caso de Asociación no se podía hacer de la misma manera.

El problema principal que tenía era que no me dejaba sobre escribir el método del constructor de Element, cosa que me imposibilitaba añadir la url nueva pedida con el factory. Es por ello por lo que la solución óptima que he visto ha sido hacer que Asociación a parte de heredar de Element también implementase de una nueva Interfaz que defino como WebElementInterface, que incluye un get y un set de la url. Luego en Factory, creo una clase abstracta llamada WebElementFactory de la que ahora pasa a extender AsociacionFactory, que incluye un constructor con la nueva url. Por último, en AsociacionCommandController,

la clase responsable de crear nuevas Asociaciones, sobrescribo los métodos de `ElementBaseCommandController` de `post()` y `put()` para que me permite poner la url.

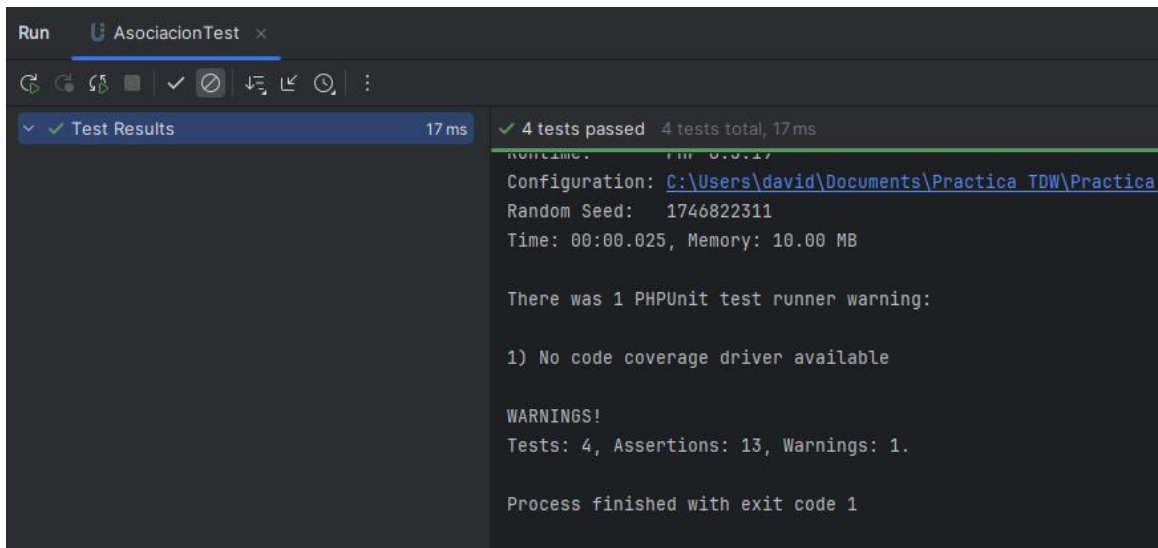
De esta manera, consigo que me permita poner el nuevo atributo url en Asociaciones.

Una vez hecho esto, configuro las rutas en un nuevo archivo en config llamado `routesAsociaciones` donde asocio cada ruta con su controlador correspondiente. También especifico para aquellas operaciones que requiera un scope de writer como en `post`, `put` y `delete` que requieran una autenticación JWT middleware.

Por último, añado al Bootstrap la dirección de `routesAsociaciones`.

Tests realizados

Solo he tenido tiempo de implementar los tests de Entity de Asociacion. Aquí verifico que el comportamiento interno de la clase Asociación.



```
Run AsociacionTest x
Test Results 17 ms
✓ 4 tests passed 4 tests total, 17 ms
Configuration: C:\Users\david\Documents\Practica TDW\Practica
Random Seed: 1746822311
Time: 00:00.025, Memory: 10.00 MB

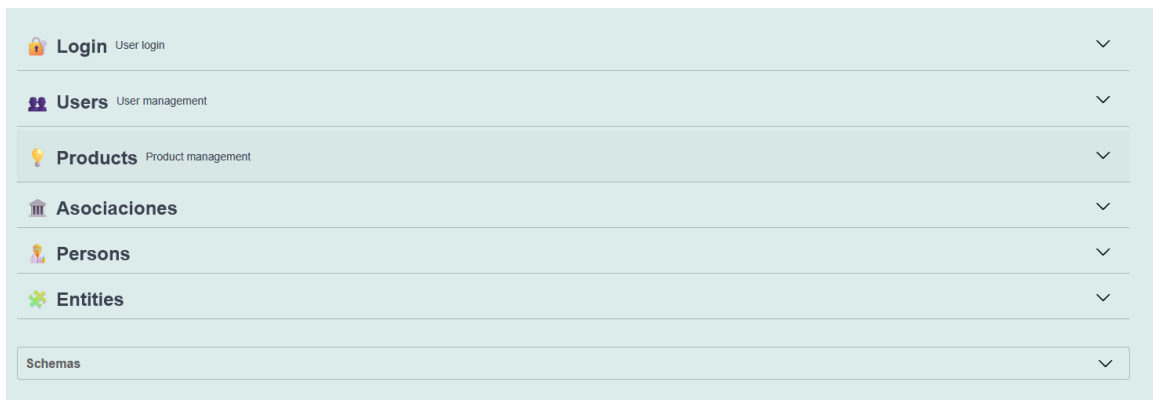
There was 1 PHPUnit test runner warning:

1) No code coverage driver available

WARNINGS!
Tests: 4, Assertions: 13, Warnings: 1.

Process finished with exit code 1
```

OpenApi



He creado nuevos paths para Asociaciones, Persons y Entities.

He tenido algún problema por olvidarme de poner en las rutas que necesitaban Middleware el bloque de:

```
security:
  - MiWApiSecurity:
    - reader
    - writer
```

Esto me provocaba que me saliese siempre un aviso de 401 “No Authorization Header” y 403 de “Forbidden: You don’t have permission”, a pesar de haber hecho el Authorize para recibir el Access token.