Interfaz RESTful UDC

Autor: Javier Martínez Fernández

I. Introducción al proyecto

Para poder implementar una interfaz REST que acceda a las operaciones de diversas entidades desde el front-end tal y como se pide para el trabajo extra de matrícula de honor, se ha implementado una API mediante el framework para php de Laravel y un paquete instalable de Swagger. De esta forma he conseguido, mediante anotaciones Swagger desde el propio framework de Laravel, tener una interfaz interactiva en el front-end a través de la cual se pueden acceder a todas las operaciones implementadas en la API.

II. Documentación detallada de la API

Base path: /api (En nuestro ejemplo de prueba: http://localhost:8000/api. Para acceder a la interfaz interactiva: http://localhost:8000/api/documentation. Aparecerá un link en la pantalla de login de la web UDC para acceder a dicha interfaz interactiva)

Recurso User

HTTP	URI	DESCRIPCIÓN	CÓDIGOS DE ESTADO
POST	/user	Crea un nuevo usuario en la aplicación. Para ello se deberá pasar en el cuerpo de la llamada el tipo User en formato JSON, tal y como se muestra en el ejemplo.	Si el usuario ha sido creado correctamente, devuelve “201 Created”: “User created succesfully. For user info: GET user/{nickUsuario}/{passUsuario}”. Si el usuario ya existe en la aplicación, devuelve “400 Bad Request”: “User already exists”. Si la creación del usuario contiene errores de validación, devuelve “400 Bad Request” junto con los correspondientes errores de validación.

POST	/user/{dniUsuario}/{nickJugador}/{idVideojuego}	Crea un nuevo seguimiento entre el usuario con dni = dniUsuario y el jugador con nick = nickJugador e id = idVideojuego. Al crear un nuevo seguimiento se verá reflejado en la web un nuevo seguimiento en la vista "Mis Seguimientos" . El idVideojuego lo podremos consultar en la llamada correspondiente a la consulta de jugadores.	Si el seguimiento ha sido creado correctamente, devuelve "201 Created": "Action successfully". Si el usuario no ha sido encontrado, devuelve "404 Not Found": "User not found". Si el jugador no ha sido encontrado, devuelve "404 Not Found": "Player not found".
GET	/user/{nickUsuario}/{passUsuario}	Consulta los datos correspondientes al usuario con nick = nickUsuario y pass = passUsuario. La respuesta se devolverá en formato JSON, tal y como se muestra en el ejemplo.	Si la consulta ha sido satisfactoria, devuelve "200 Ok" junto con la respuesta en formato JSON. Si el usuario no ha sido encontrado, devuelve "404 Not Found": "User not found".
GET	/user/{dniUsuario}	Consulta todos los seguimientos para el usuario con dni = dniUsuario. La respuesta se devolverá en formato JSON, tal y como se muestra en el ejemplo. Podremos utilizar el parámetro tipo query "search" para buscar un jugador por nombre. También podremos utilizar el parámetro tipo query "order", cuyo valor "alfabetic" sirve para ordenar los jugadores alfabéticamente y cuyo valor "-alfabetic" es el orden inverso al anterior.	Si la consulta ha sido satisfactoria, devuelve "200 Ok" junto con la respuesta en formato JSON. Si el parámetro tipo query "order" es incorrecto, devuelve "400 Bad Request": "Incorrect value for order".
DELETE	user/{dniUsuario}/{nickJugador}/{idVideojuego}	Elimina el seguimiento entre el usuario con dni = dniUsuario y el jugador con nick = nickJugador e id = idVideojuego. En la web desaparecerá dicho seguimiento de la vista	Si el seguimiento ha sido eliminado correctamente, devuelve "201 Deleted":"Delete successfully". Si el usuario no ha sido encontrado, devuelve

		“Mis Seguimientos”.	“404 Not Found”: “User not found”. Si el jugador no ha sido encontrado, devuelve “404 Not Found”: “Player not found”.
POST	user/{opinion}/{dniUsuario}/{nickJugador}/{idVideojuego}	Crea un nuevo comentario = opinion del usuario con dni = dniUsuario para el jugador con nick = nickJugador e id = idVideojuego. En la web esto se verá reflejado con dicho comentario en el seguimiento del jugador correspondiente.	Si el comentario ha sido creado correctamente, devuelve “201 Created”: “Action successfully”. Si el comentario contiene errores de validación, devuelve “400 Bad Request” junto con los correspondientes errores de validación. Si el usuario no ha sido encontrado, devuelve “404 Not Found”: “User not found”. Si el jugador no ha sido encontrado, devuelve “404 Not Found”: “Player not found”.
PUT	user/{dniUsuario}/{newPass}	Actualiza la contraseña del usuario con dni = dniUsuario, siendo la nueva contraseña = newPass. En la web se verá reflejado en el perfil del usuario.	Si la contraseña ha sido actualizada correctamente, devuelve “201 Updated”: “Pass update successfully”. Si la contraseña contiene errores de validación, devuelve “400 Bad Request” junto con los correspondientes errores de validación.
PUT	user/{dniUsuario}/{nombreUsuario}/{nickUsuario}/{em	Actualiza el nombre = nombreUsuario, nick = nickUsuario, email = emailUsuario y teléfono = telefonoUsuario, del usuario con dni = dniUsuario. En la web se verá reflejado en el perfil	Si el perfil ha sido actualizado correctamente, devuelve “201 Updated”: “Profile update successfully”. Si contiene errores de validación, devuelve “400

	ailUsuario}/{telefonoUsuario}	del usuario.	Bad Request" junto con los correspondientes errores de validación.

Ejemplo JSON del cuerpo para la llamada POST /user:

```
{
  "dniUsuario": "64767587G",
  "nombreCompletoUsuario": "Pepe de la Torre",
  "nickUsuario": "Pepexxii",
  "emailUsuario": "pepexii@gmail.com",
  "fechaNacimientoUsuario": "2002-08-12",
  "numTelefonoUsuario": "645576789",
  "passUsuario": "Constantina2020",
  "confirmPassUsuario": "Constantina2020"
}
```

Ejemplo JSON de la respuesta para la llamada GET /user/{nickUsuario}/{passUsuario}:

```
{
  "DNI": "64767587G",
  "NOMBRE": "Pepe de la Torre",
  "NICK": "Pepexxii",
  "EMAIL": "pepexii@gmail.com",
  "FECHA_NACIMIENTO": "2002-08-12",
  "TELEFONO": "645576789",
  "CONTRASEÑA": "Constantina2020"
}
```

Ejemplo JSON de la respuesta para la llamada GET /user/{dniUsuario}:
(number)= se refiere al idVideojuego

```
[
  "luchoy(1): El mejor jugador del mundo",
  "albdom(1): Buen partido el de ayer",
  "loicab(1): -You are the best",
  "manubri(1): toma toma que bien jugado"
]
```

El formato de los campos para POST deberá de ser el mismo que los del ejemplo.

Recurso Jugadores

HTTP	URI	DESCRIPCIÓN	CÓDIGOS DE ESTADO
GET	/jugadores	Consulta todos los jugadores del club. Si utilizamos el parámetro tipo query “bests” con valor “true”, nos mostrará los mejores jugadores de cada una de las líneas de videojuegos del club. Podremos utilizar el parámetro tipo query “search” para buscar un jugador por nombre. También podremos utilizar el parámetro tipo query “order”, cuyo valor “alfabetic” sirve para ordenar los jugadores alfabéticamente y cuyo valor “-alfabetic” es el orden inverso al anterior.	Si la consulta ha sido satisfactoria, devuelve “200 Ok” junto con la respuesta en formato JSON. Si el parámetro tipo query “order” es incorrecto, devuelve “400 Bad Request”: “Incorrect value for order”.

Ejemplo JSON de la respuesta para la llamada GET /jugadores?bests=true:


```
[
  "samuellin(8)",
  "angba(2)",
  "thalm22(5)",
  "nicoldom(3)",
  "hectdelafu(5)",
  "antoinerelad(9)",
  "luchoy(1)",
  "luciore(2)",
  "dominguitomg(7)",
  "gugime123(4)",
  "mauro2sant(3)",
  "hoyosmaster(10)",
  "fierro(8)",
  "lucober(2)",
  "pagal(2)"
]
```

III. Memoria del proyecto

Para la realización de este proyecto he utilizado el framework Laravel como se ha comentado en la introducción. Laravel tiene como objetivo ser un framework que permita el uso de una sintaxis elegante y expresiva para crear código de forma sencilla y permitiendo multitud de funcionalidades. Intenta aprovechar lo mejor de otros frameworks y aprovechar las características de las últimas versiones de PHP.

Al utilizar Laravel he considerado oportuno crear un nuevo proyecto a parte del proyecto enfocado a la web de IISSI2, ya que para que la API funcione se tendrá que desplegar desde la línea de comandos un servidor para dicho proyecto Laravel (comando para dicho despliegue: "php artisan serve"). En este proyecto Laravel he incluido todos los scripts necesarios de la web de UDC para poder implementar las funciones que modifiquen la base de datos.

Me gustaría destacar que las funciones implementadas en la API son las que he considerado más importantes con respecto al trabajo de este último cuatrimestre, ya que en el caso de nuestro trabajo tenemos una gran cantidad de funciones referidas a la base de datos y todas serían demasiados casos para lo que se pedía en este trabajo extra. Se han implementado los cuatro tipos de funciones requeridas (GET, POST, PUT, DELETE), asegurándome de que estas funciones cumplan con los principios básicos de una interfaz RESTful, como puede ser el principio de idempotencia. Al implementar dicha API he tenido que hacer pequeñas modificaciones en primitivas de acceso a entidades de nuestra base de datos, además de haberme visto obligado a crear alguna primitiva nueva para ofrecer una mejor experiencia con la interfaz RESTful. En los scripts referidos a las primitivas de acceso a la base de datos del proyecto Laravel solo he incluido aquellas funciones que he necesitado para implementar la API. Soy consciente de que se pueden mejorar aspectos de seguridad y otros aspectos para las llamadas a la API, pero por falta de tiempo y ya que creo que he demostrado adecuadamente que se hacer lo que se pedía, pues no he podido.

En cuanto a la implementación de la interfaz interactiva mediante Swagger, he utilizado un paquete para dicho proyecto Laravel llamado darkaonline/l5-swagger, el cual permite mediante anotaciones en el propio proyecto Laravel definir la documentación de Swagger y construir la interfaz. Para poder utilizar dicho paquete correctamente he tenido que actualizar la versión de PHP 5.6 (la que teníamos para la asignatura) a la versión PHP 7.2, ya que para poder implementar Swagger con Laravel se requería la versión PHP 7.0 o superior, por lo que he tenido que hacer varios cambios en el XAMPP para cambiar la versión y que siga funcionando toda la web y demás correctamente. Obviamente para poder programar correctamente he tenido que dejar de utilizar Aptana (ya que no soporta las últimas versiones de PHP) y utilizar Visual Studio Code.

Para terminar, me gustaría destacar la gran labor de búsqueda que he tenido que hacer para asimilar dicho software que es nuevo para mí. Esta labor de búsqueda es lo que más tiempo me ha llevado para implementar correctamente todo, ya que una vez lo había asimilado y entendido solo quedaba programar.

Para acceder a dicha interfaz interactiva implementada mediante Swagger, he adjuntado un link en la zona de login de la web de UDC. Además dicha ruta está plasmada en el punto II de este documento.

Gracias.