Gestión de la Información en la Web
Curso 2024-25
Práctica 7 – API REST

Fecha de entrega: domingo 17 de noviembre 2024, 23:55h

Entrega de la práctica
La entrega de la práctica se realizará a través del Campus Virtual de la asignatura mediante un fichero `pr7.py`. El esqueleto de este fichero se puede descargar del Campus Virtual.

Lenguaje de programación
Python 3.11 o superior.

Calificación
Se medirá la corrección mediante tests de unidad. Además de la corrección, se valorará la **calidad, concisión y claridad del código**, la incorporación de **comentarios** explicativos, su **eficiencia** tanto en tiempo como en memoria y la puntuación obtenida en Pylint.

Declaración de autoría e integridad
Todos los ficheros entregados contendrán una cabecera en la que se indique la asignatura, la práctica, el grupo y los autores. Esta cabecera también contendrá la siguiente declaración de integridad:

> *Declaramos que esta solución es fruto exclusivamente de nuestro trabajo personal. No hemos sido ayudados por ninguna otra persona o sistema automático ni hemos obtenido la solución de fuentes externas, y tampoco hemos compartido nuestra solución con otras personas de manera directa o indirecta. Declaramos además que no hemos realizado de manera deshonesta ninguna otra actividad que pueda mejorar nuestros resultados ni perjudicar los resultados de los demás.*

No se corregirá ningún fichero que no venga acompañado de dicha cabecera.

En esta práctica implementaremos una API REST muy pequeña que nos permitirá operar con asignaturas: añadirlas, consultarlas, editarlas y borrarlas. Este servicio debe implementarse en **Flask** y deberá escuchar en el **puerto 5000**.

Para realizar la práctica debéis descargar el esqueleto del servicio web que está en el Campus Virtual y usarlo como base. Este fichero incluye el código básico de un servidor Flask y únicamente en necesario definir e implementar las distintas rutas.

No es necesario que el servicio disponga de *persistencia*, es decir, podéis considerar que cada vez que se arranca éste comienza con un estado vacío sin ninguna asignatura. Si aún así queréis obtener persistencia podéis utilizar ficheros (de texto o binarios) o conectar directamente con una base de datos (SQLite por ejemplo). En la próxima práctica se tratará cómo obtener persistencia usando una traducción automática entre Python y MongoDB.

1. Esquema de las asignaturas

En nuestra API representaremos las asignaturas únicamente mediante JSON con los siguientes campos y tipos:

```
{
  "id": int,
  "nombre": str,
  "numero_alumnos": int,
  "horario: [
            {
              "dia": str,
              "hora_inicio": int,
              "hora_final": int
            }
          ]
}
```

Nótese que `horario` es una lista de cero, uno o varios documentos JSON, donde cada uno de estos documentos contendrá tres campos (`dia`, `hora_inicio` y `hora_final`). El campo `id` es incluido automáticamente por la API al recibir asignaturas, es decir, `id` **no debe aparecer en las asignaturas que se envían para añadir o reemplazar**, pero el servidor sí que lo incluirá en las respuestas cuando el usuario consulte una determinada asignatura.

Por simplicidad, supondremos que los identificadores son números naturales incrementales comenzando en 0.

2. Rutas de la API REST

A continuación se explican las distintas rutas a las que deberá atender el API REST. Para cada una de ellas se incluyen varios **ejemplos**[1] usando `curl`, el comando Linux para realizar peticiones desde el terminal y que se usa comúnmente para documentar este tipo de API, aunque para realizar estas mismas peticiones también podréis utilizar extensiones del navegador como «Advanced REST client» o «RESTED», e incluso aplicaciones con interfaz gráfica como «Insomnia» o «Postman». Los parámetros de `curl` que se usan son los que detallamos a continuación y la salida producida por `curl` es todo lo que aparece después del comando, comenzando con «`HTTP/1.0`»:

- `-i`: para que la salida muestre las cabeceras de la contestación

[1] Si los ejecutáis en un terminal Linux funcionarán, pero no es seguro que sirvan en el terminal de Windows (CMD o PowerShell).

- `-X`: indica el método HTTP
- `--header`: contenido de la cabecera de la petición
- `--data`: contenido del cuerpo de la petición

2.1. /asignaturas

Mediante esta ruta se podrá añadir una nueva asignatura, borrar todas las asignaturas y consultar las asignaturas:

- `DELETE`: Elimina todas las asignaturas que existan, devolviendo el código «**204 No Content**». Ejemplo:

```
$ curl -i -X "DELETE" http://localhost:5000/asignaturas
HTTP/1.0 204 NO CONTENT
Content-Type: text/html; charset=utf-8
Server: Werkzeug/3.0.4 Python/3.11.10
Date: Mon, 02 Nov 2024 18:43:05 GMT
```

- `POST`: Añade una nueva asignatura. Se debe incluir en la petición la asignatura a incluir representada como JSON sin ningún campo `id`. Si la asignatura no cumple con el esquema o tiene tipos incorrectos en los campos, se devolverá el código «**400 Bad Request**», en otro caso se devolverá el código «**201 Created**» y el contenido «`{"id": N}`» donde N es el identificador único asignado a la asignatura que se acaba de crear. Ejemplos

```
$ curl -i -X POST --header "Content-Type: application/json" --data '{"nombre": "GIW",
"numero_alumnos": 40, "horario": [{"dia": "martes", "hora_inicio": 14,
"hora_final": 16}] }' http://localhost:5000/asignaturas
HTTP/1.0 201 CREATED
Content-Type: application/json
Content-Length: 17
Server: Werkzeug/3.0.4 Python/3.11.10
Date: Mon, 02 Nov 2024 18:46:55 GMT

{
  "id": 1432
}

$ curl -i -X POST --header "Content-Type: application/json" --data
'{"asignatura": "mal formada"}' http://localhost:5000/asignaturas
HTTP/1.0 400 BAD REQUEST
Content-Type: text/html; charset=utf-8
Content-Length: 0
Server: Werkzeug/3.0.4 Python/3.11.10
Date: Mon, 02 Nov 2024 18:51:50 GMT
```

- `GET`: permite obtener las URL de las asignaturas. Si no se pasa ningún parámetro, se devolverá un JSON `{"asignaturas": L}` donde L es una lista de URL para acceder a las distintas asignaturas. Ejemplo:

```
$ curl -i -X GET http://localhost:5000/asignaturas
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 77
```

```
Server: Werkzeug/3.0.4 Python/3.11.10
Date: Mon, 02 Nov 2024 18:53:15 GMT

{
  "asignaturas": [
    "/asignaturas/1431",
    "/asignaturas/1432",
    "/asignaturas/1433",
    "/asignaturas/1434"
  ]
}
```

Esta ruta también admite los parámetros `page` y `per_page` (si aparece uno siempre debe aparecer el otro) para indicar el número de página solicitada (la primera página es la página **1**) y el número de asignaturas a devolver en cada página, respectivamente. También admite el parámetro `alumnos_gte` para devolver únicamente aquellas asignaturas que tienen ese número de alumnos o más. **Tened en cuenta que pueden aparecer los 3 parámetros a la vez, por lo que habrá que realizar el paginado a los resultados obtenidos del filtrado por número de alumnos**. Si los parámetros utilizado en la petición son incorrectos, se devolverá «**400 Bad Request**», si el listado que se devuelve contiene todas las asignaturas existentes se devolverá «**200 OK**» y si no están todas las asignaturas en el resultado se devolverá «**206 Partial Content**». Ejemplos:

```
$ curl -i -X GET "http://localhost:5000/asignaturas?page=1"
HTTP/1.0 400 BAD REQUEST
Content-Type: text/html; charset=utf-8
Content-Length: 0
Server: Werkzeug/3.0.4 Python/3.11.10
Date: Mon, 02 Nov 2024 19:06:13 GMT

$ curl -i -X GET "http://localhost:5000/asignaturas?page=1&per_page=1"
HTTP/1.0 206 PARTIAL CONTENT
Content-Type: application/json
Content-Length: 51
Server: Werkzeug/3.0.4 Python/3.11.10
Date: Mon, 02 Nov 2024 19:06:54 GMT

{
  "asignaturas": [
    "/asignaturas/1431"
  ]
}

$ curl -i -X GET "http://localhost:5000/asignaturas?page=1&per_page=5"
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 77
Server: Werkzeug/3.0.4 Python/3.11.10
Date: Mon, 02 Nov 2024 19:07:19 GMT

{
  "asignaturas": [
    "/asignaturas/1431",
    "/asignaturas/1432",
    "/asignaturas/1433",
```

```
    "/asignaturas/1434"
  ]
}
```

2.2. /asignaturas/X

Mediante esta ruta se podrá consultar una nueva asignatura, borrarla o modificarla:

- `DELETE`: borra una determinada asignatura. Si existe devuelve «**204 No Content**» y si no existe devuelve «**404 Not Found**». Ejemplo:

```
$ curl -i -X DELETE "http://localhost:5000/asignaturas/1431"
HTTP/1.0 204 NO CONTENT
Content-Type: text/html; charset=utf-8
Server: Werkzeug/3.0.4 Python/3.11.10
Date: Mon, 02 Nov 2024 19:12:17 GMT

$ curl -i -X DELETE "http://localhost:5000/asignaturas/9999"
HTTP/1.0 404 NOT FOUND
Content-Type: text/html; charset=utf-8
Content-Length: 0
Server: Werkzeug/3.0.4 Python/3.11.10
Date: Mon, 02 Nov 2024 19:12:21 GMT
```

- `GET`: Devuelve los datos de una asignatura concreta con código «**200 OK**» si existe, o «**404 Not Found**» si no existe. Ejemplo:

```
$ curl -i -X GET "http://localhost:5000/asignaturas/1433"
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 168
Server: Werkzeug/3.0.4 Python/3.11.10
Date: Mon, 02 Nov 2024 19:15:11 GMT

{
  "horario": [
    {
      "dia": "martes",
      "hora_final": 16,
      "hora_inicio": 14
    }
  ],
  "id": 1433,
  "nombre": "GIW",
  "numero_alumnos": 40
}
```

- `PUT`: Reemplaza completamente una asignatura que existe usando los datos que se pasan en la petición, **pero conservando su identificador**. Si la asignatura no existe se devolverá «**404 Not Found**», y si existe se devolverá «**200 OK**». La asignatura que se pasa en la petición debe cumplir el esquema o se devolverá el código «**400 Bad Request**». Ejemplos:

```
$ curl -i -X PUT --header "Content-Type: application/json" --data '{"nombre": "GIW",
"numero_alumnos": 40, "horario": [{"dia": "martes", "hora_inicio": 14,
```

```
"hora_final": 16}] }' http://localhost:5000/asignaturas/9999
HTTP/1.0 404 NOT FOUND
Content-Type: text/html; charset=utf-8
Content-Length: 0
Server: Werkzeug/3.0.4 Python/3.11.10
Date: Mon, 02 Nov 2024 19:19:49 GMT

$ curl -i -X PUT --header "Content-Type: application/json" --data '{"nombre": "ALP",
"numero_alumnos": 22, "horario": [] }' http://localhost:5000/asignaturas/1433
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 0
Server: Werkzeug/3.0.4 Python/3.11.10
Date: Mon, 02 Nov 2024 19:20:19 GMT

$ curl -i -X PUT --header "Content-Type: application/json" --data '{"nombre": "ALP",
"numero_alumnos": 22}' http://localhost:5000/asignaturas/1433
HTTP/1.0 400 BAD REQUEST
Content-Type: text/html; charset=utf-8
Content-Length: 0
Server: Werkzeug/3.0.4 Python/3.11.10
Date: Mon, 02 Nov 2024 19:20:31 GMT
```

- `PATCH`: Actualiza exactamente un campo de una asignatura que existe. Por lo tanto, en la petición se debe pasar un JSON con **un único campo** y que deber ser **válido** según el esquema de las asignaturas. Si la asignatura no existe se devolverá «**404 Not Found**», y si existe se devolverá «**200 OK**». Si el JSON pasado no es válido se devolverá «**400 Bad Request**». Ejemplos:

```
$ curl -i -X PATCH --header "Content-Type: application/json"
--data '{"numero_alumnos": 1000}' http://localhost:5000/asignaturas/1433
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 0
Server: Werkzeug/3.0.4 Python/3.11.10
Date: Mon, 02 Nov 2024 19:25:04 GMT

$ curl -i -X PATCH --header "Content-Type: application/json"
--data '{"numero_alumnos": "tres"}' http://localhost:5000/asignaturas/1433
HTTP/1.0 400 BAD REQUEST
Content-Type: text/html; charset=utf-8
Content-Length: 0
Server: Werkzeug/3.0.4 Python/3.11.10
Date: Mon, 02 Nov 2024 19:26:06 GMT
```

2.3. `/asignaturas/X/horario`

En esta ruta se puede obtener directamente el horario de una asignatura concreta.

- `GET`: Devuelve un JSON con un único campo `horario` con el horario de la asignatura en cuestión. Si esta asignatura existe se devolverá el código «**200 OK**» y si no «**404 Not Found**». Ejemplos:

```
$ curl -i -X GET http://localhost:5000/asignaturas/9999/horario
HTTP/1.0 404 NOT FOUND
Content-Type: text/html; charset=utf-8
Content-Length: 0
```

```
Server: Werkzeug/3.0.4 Python/3.11.10
Date: Mon, 02 Nov 2024 19:29:55 GMT

$ curl -i -X GET http://localhost:5000/asignaturas/1434/horario
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 108
Server: Werkzeug/3.0.4 Python/3.11.10
Date: Mon, 02 Nov 2024 19:29:59 GMT

{
  "horario": [
    {
      "dia": "martes",
      "hora_final": 16,
      "hora_inicio": 14
    }
  ]
}
```

3. Tests de unidad

Manejar correctamente las distintas rutas implica tener en cuenta muchos matices, entre los que destaca detectar que la petición recibida es correcta y decidir el código de estado a devolver. Para que podáis comprobar con facilidad que vuestra API cumple con los requerimientos, he creado una serie de tests de unidad que realizan peticiones a la API en `http://127.0.0.1:5000` y comprueban que los resultados son los esperados. Todos los tests están incluidos en el fichero `tests.py` que os podéis descargar del Campus Virtual.

Los tests de unidad están realizados con la biblioteca estándar `unittest`, así que no tendréis que instalar nada adicional para poder lanzarlos. Para ejecutarlos debéis abrir un terminal en el directorio donde esté el fichero `tests.py` y escribir lo siguiente:

- Para lanzar **todos** los tests que existen:

```
$ python -m unittest tests.TestREST
```

- Para lanzar **un test concreto**, por ejemplo `test_get_horario_inexistente()`:

```
$ python -m unittest tests.TestREST.test_get_horario_inexistente
```

Importante: los test de unidad suponen que vuestra API REST se está ejecutando y escuchando en `http://127.0.0.1:5000`, que es el valor por defecto en Flask. En total, el tiempo necesario para ejecutar todos los tests de unidad no debería exceder de 5-10 segundos.