Programación 2 > módulo 3

>Tecnicatura Universitaria en Desarrollo de Software

UPATECO UNIVERSIDAD PROVINCIAL
ADMINISTRACIÓN • TECNOLOGÍA • OFICIOS

Ministerio de Economía
y Servicios Públicos
Gobierno de Salta

academia
CIMNE IBER

módulo 3

Tecnicatura Universitaria en Desarrollo de Software

Programación 2

Módulo 3: Backend

Tema 1: Routing

UPATECO UNIVERSIDAD PROVINCIAL ADMINISTRACIÓN • TECNOLOGÍA • OFICIOS

Ministerio de Economía y Servicios Públicos Gobierno de Salta

academia CIMNE IBER

Trabajo Práctico 1 - Routing

El objetivo de este trabajo es poner en práctica los fundamentos a cerca de **routing** mediante el uso del framework Flask.

Debe emplearse la estructura de proyecto vista en clases para resolver los ejercicios.

Estructura de proyecto

```
/../mi_proyecto/
|--hola_mundo/
|    |-- static/
|    |-- __init__.py
|    config.py
|    run.py
```

Consideraciones generales

1. Durante este trabajo práctico todos los endpoints deberán ser definidos en el archivo `__init__.py`, particularmente en la *application factory*. Sin embargo, las funciones asociadas a cada endpoint pueden ser definidas en otro archivo y luego importadas en el archivo `__init__.py`. Posteriormente, veremos cómo administrar las rutas de manera más eficiente.

2. Las **respuestas** de cada endpoint deben ser en formato JSON.

3. Algunos ejercicios pueden producirse errores en base a los datos provistos en una solicitud. Sin embargo, no se ha visto cómo manejarlos de manera adecuada. Por el momento, nos conformaremos con mostrar un mensaje de error en la respuesta de la solicitud. Es decir, si se produce un error, devolveremos un mensaje de error en formato JSON. Por ejemplo:

```
{
    'error': 'Ha ocurrido un error'
}
```

U otra variante de mensaje de error que se considere adecuada.

4. Las respuestas de cada endpoint deben incluir el código de estado HTTP `200` (*OK*) en caso de que la solicitud se haya procesado correctamente. En caso contrario, se debe devolver el código de estado HTTP `400` (*Bad Request*). Por ejemplo, si se produce un error, la sentencia `return` de la función asociada al endpoint debería ser algo como:

   ```
   return {'error': 'Ha ocurrido un error'}, 400
   ```

   Se debe realizar una tarea similar para el caso análogo, es decir, cuando la solicitud se procesa correctamente.

5. Algunos ejercicios pueden requerir el uso de archivos complementarios (no necesariamente de código), los cuales se encuentran adjuntos a este trabajo práctico. Según la estructura de proyecto propuesta, estos archivos deberán ser almacenados en el directorio `/static`.

Ejercicio 1

Crear una aplicación Flask donde se defina un endpoint para la ruta `/`, la cual debe mostrar un mensaje de bienvenida a quien envíe una petición a dicho endpoint. Por ejemplo, podemos mostrar el mensaje `Bienvenidx!`.

Ejercicio 2

Crear una aplicación Flask que tenga una ruta para el endpoint `/info` que muestre un mensaje de bienvenida en el que se indique el nombre de la aplicación. Para esto deberá usarse la clase `Config`, donde uno de sus atributos de clase (al cual podemos nombrar `APP_NAME`) será justamente el nombre de la app. Como resultado tendríamos que devolver algo como `Bienvenidx a Routing App`.

> Recordemos modificar los atributos a la clase `Config` para resolver este ejercicio.

Ejercicio 3

Definir un endpoint para la ruta `/about` que muestre información sobre la aplicación en formato JSON:

- El nombre de la app.
- Descripción de la app.

- Una lista con los desarrolladores. Cada desarrollador debe tener:
  - Nombre
  - Apellido
- La versión de la app.

Por ejemplo:

```
{
    'app_name':'Routing App',
    'description': 'Aplicación para practicar routing en Flask',
    'developers': [
      {
      'nombre': 'Carlos',
      'apellido': 'Santana'
      },
      {
      'nombre': 'James',
      'apellido': 'Hetfield'
      }
      ],
    'version': '1.0.0'
}
```

> Todos estos datos pueden ser almacenados también en la clase `Config` y luego acceder a ellos mediante el atributo de clase correspondiente.

Ejercicio 4

Definir un endpoint para la ruta `/sum/<int:num1>/<int:num2>` que sume dos números enteros y muestre el resultado. Por ejemplo, para la petición `/sum/20/10` devolveríamos `30`.

Ejercicio 5

Se solicita definir un endpoint que calcule la edad de una persona en base a su fecha de nacimiento. Para ello se establece la ruta `/age/<dob>`, donde `dob` se refiere a *day of birth* y se encuentra en formato ISO 8601 (YYYY-MM-DD).

Por ejemplo, para la petición HTTP `/age/2001-10-20` y suponiendo que es enviada en la fecha `2023-05-16` debería devolver 21.

En caso de que la fecha de nacimiento sea posterior a la fecha actual, se debe devolver un mensaje de error en formato JSON.

> Ayuda: puede hacer uso del módulo `date` del paquete `datetime` para realizar el cálculo de edad.

Ejercicio 6

Se desea realizar operaciones matemáticas simples mediante nuestra API. Para ello se solicita definir un endpoint para `/operate/<string:operation>/<int:num1>/<int:num2>` que opere dos números enteros y devuelva el resultado de la operación como respuesta a la petición. El parámetro de ruta `operation` indica la operación que se desea realizar, y puede tomar los siguientes valores:

- `sum` para sumar los números.
- `sub` para restar los números.
- `mult` para multiplicar los números.
- `div` para dividir los números. Si el parámetro ruta `num2` es `0`, el cual representa al divisor, mostraremos un mensaje indicando que la división no está definida para esos valores.
- En cualquier otro caso, mostraremos un mensaje indicando que no existe una ruta definida para ese *enpoint*.

Ejercicio 7

Reformular el ejercicio anterior para el endpoint `/operate`, con la diferencia que esta ruta deberá recibir parámetros de consulta (`query params`) `operation`, `num1` y `num2` en lugar de parámetros de ruta como veíamos en el ejercicio 8.

Por ejemplo, para la petición `/operate?operation=sum&num1=20&num2=10` devolveríamos `30` en nuestra respuesta.

Ejercicio 8

Se solicita crear una ruta para en el endpoint `/title/<string:word>`, el cual aplica el formato título al parámetro de ruta `word` (la primera letra de la palabra es mayúscula y el resto minúscula), y devolverá una respuesta en formato JSON donde la palabra formateada estará asociada a la clave `formatted_word`. Por ejemplo, para la petición `/title/SARmienTo` devolverá:

```
{
    "formatted_word": "Sarmiento"
}
```

Ejercicio 9

Para en el endpoint `/formatted/<string:dni>` se necesita implementar una función que reciba como parámetro de ruta un DNI, y devuelva el DNI convertido a un entero. El parámetro de ruta recibido puede contar con las siguientes características a ser tenidas en cuenta para la conversión:

1. El DNI siempre tendrá 8 caracteres numéricos en la cadena recibida. Por ejemplo, `23456007`, `23.456.007`, `23-456-007`, `00023456`, etc.
2. El DNI puede contener puntos (`.`) como separador de miles.
3. El DNI puede contener guiones (`-`) como separador de miles.
4. El DNI puede tener ceros a la izquierda. Por ejemplo, `00023456`. En este caso el DNI no se considera válido, por lo que se deberá devolver un mensaje de error en la respuesta.

La respuesta a una petición debe tener formato JSON, donde el DNI formateado estará asociado a la clave `formatted_dni`. Por ejemplo, para la petición `/formatted/23.456.007` devolverá:

```
{
    "formatted_dni": 23456007
}
```

Ejercicio 10

Definir un endpoint para la ruta `/format`, donde recibirá los datos de un usuario como parámetros de consulta (`query params`):

- `firstname`: nombre del usuario. Suponemos que tiene un único nombre.
- `lastname`: apellido del usuario
- `dob`: fecha de nacimiento.
- `dni`: documento nacional de identidad.

Deberemos procesar los datos recibidos en la petición y devolver una respuesta en formato JSON de manera tal que se cumplan las siguientes condiciones:

1. El nombre y apellido deben tener únicamente la primera letra de cada palabra en mayúscula y el resto en minúscula.
2. Debe calcularse la edad de la persona en base a su fecha de nacimiento y devolver dicho valor.
3. El DNI es recibido como una cadena de caracteres, y debe ser formateado de manera tal que se eliminen los caracteres especiales como puntos (`.`) o guiones (`-`), y se convierta a un entero.

> Si el DNI recibido no es válido, es decir, no tiene 8 caracteres numéricos, se debe devolver un mensaje de error en la respuesta. Si la fecha de nacimiento recibida es posterior a la fecha actual, se debe devolver un mensaje de error en la respuesta.

Por ejemplo, para la petición con el endpoint `/format?firstname=LUiS&lastname=JUARez&dob=2001-08-27&dni=44.010.777`, suponiendo que la petición se solicitó en la fecha `2023-05-19` deberíamos devolver:

```
{
      "firstname": "Luis",
      "lastname": "Juarez",
      "age": 21,
      "dni": 44010777
}
```

Ejercicio 11

Se envían palabras clave a nuestra API, y se solicita encriptarlas y devolver la palabra en código morse. Razón por la cual se deberá definir un endpoint para `/encode/<string:keyword>`.

Se cuenta con las siguientes tablas de referencia:

Letras	Código	Letras	Código
A	._	N	_.
B	_...	O	___
C	_._.	P	.__.
D	_..	Q	__._
E	.	R	._.
F	.._.	S	...
G	__.	T	_

Letras	Código	Letras	Código
H	....	U	..-
I	..	V	...-
J	.---	W	.--
K	-.-	X	-..-
L	.-..	Y	-.--
M	--	Z	--..

Números	Código	Números	Código
1	.----	6	-....
2	..---	7	--...
3	...--	8	---..
4	....-	9	----.
5	.....	0	-----

Estos datos se encuentran registrados en un archivo adjunto llamado `morse_code.json`:

```
{
      "letters": {
      "A": ".-", "B": "-...", "C": "-.-.", "D": "-..", "E": ".", "F": "..-.", "G": "--.","H": "....", "I": "..", "J": ".---", "K": "-.-", "L": ".-..", "M": "--", "N": "-.", "O": "---", "P": ".--.", "Q": "--.-", "R": ".-.", "S": "...", "T": "-", "U": "..-",   "V": "...-", "W": ".--", "X": "-..-", "Y": "-.--", "Z": "--..", "0": "-----", "1": ".----", "2": "..---", "3": "...--", "4": "....-", "5": ".....", "6": "-....", "7": "--...", "8": "---..", "9": "----." " ": "^"
      }
}
```

Establecemos las siguientes consideraciones para casos excepcionales que pueden presentarse:

1. Los caracteres de espacio `" "` no pueden enviarse literalmente como parámetro de ruta. Por lo que, cuando queramos enviar más de una palabra clave usaremos el carácter `"^"` en su lugar. En consecuencia, las palabras clave que enviemos a nuestra API no podrán contener el carácter `"^"`.
2. Asumimos que las palabras clave que enviemos a nuestra API solo contendrán letras mayúsculas, números y el carácter `"+"` como separador de letras.

3. Las palabras clave que enviemos a nuestra API no contendrán caracteres especiales como acentos, diéresis, etc.
4. Las palabras clave que enviemos a nuestra API no contendrán espacios al principio o al final de la palabra.
5. Las palabras clave tienen un máximo de 100 caracteres en total, incluyendo el carácter `"^"` en esta cuenta.

Por ejemplo, para codificar `Buenas  tardes` tendremos que usar el endpoint `/encode/Buenas+tardes`, y como resultado obtendríamos `Buenas  tardes` en código morse: `-...+..-+.+-.+.-+...+^+-+.-+.-.+-..+.+...`.

Ejercicio 12

Se solicita definir un endpoint para realizar la operación inversa al ejercicio anterior. Es decir, se envía una palabra o frase en código morse y se solicita devolver la palabra o frase en texto plano. Para ello se define la ruta `/decode/<string:morse_code>`.

Por ejemplo, para la petición `/decode/-...+..-+.+-.+.-+...+^+-+.-+.-.+-..+.+...` devolveríamos `Buenas tardes`.

Para resolver este ejercicio podemos realizar el proceso inverso al del ejercicio anterior.

Ejercicio 13

Crear un endpoint que acepte un número en base binaria representado a través de una cadena, como puede ser, `/convert/binary/<string:num>` y devuelva el número en base decimal. Por ejemplo, para la petición `/convert/binary/1010` devolveríamos `10` en nuestra respuesta.

> El número recibido no tiene parte decimal.

Para realizar la conversión se solicita implementar el algoritmo de suma ponderada, donde cada dígito del número binario se multiplica por la potencia de 2 correspondiente a su posición, y se suman los resultados de cada multiplicación. Por ejemplo, para el número binario `1010`:

- El dígito `1` que se encuentra en la posición `3` (contando desde la derecha desde cero), por lo que se multiplica por `2^3 = 8`.
- El dígito `0` que se encuentra en la posición `2` (contando desde la derecha desde cero), por lo que se multiplica por `2^2 = 4`.

- El dígito `1` que se encuentra en la posición `1` (contando desde la derecha desde cero), por lo que se multiplica por `2^1 = 2`.
- El dígito `0` que se encuentra en la posición `0` (contando desde la derecha desde cero), por lo que se multiplica por `2^0 = 1`.

Por lo tanto, el resultado de la conversión es `8 + 0 + 2 + 0 = 10`.

La ecuación para realizar la conversión es la siguiente:

$$decimal = \sum \sum_{i=0}^{n-1} binario[i] * 2^{n-1-i}$$

Ecuación para la conversión de un número binario a decimal

Ejercicio 14

Crear un endpoint que acepte una cadena como parámetro de ruta, como puede ser, `/balance/<string:input>`. Este parámetro contendrá una serie de paréntesis, corchetes y/o llaves.

La tarea del endpoint es verificar si los símbolos están balanceados, es decir, cada símbolo de apertura tiene su correspondiente símbolo de cierre.

Para resolver este problema, se solicita utilizar la estructura de datos **pila**, y realizar el siguiente procedimiento:

1. Cada vez que se encuentre un símbolo de apertura, debe ser apilado.
2. Cada vez que se encuentre un símbolo de cierre, debe comprobarse si la cima de la pila es el símbolo de apertura correspondiente.
    - Si es así, se debe desapilar el símbolo de apertura de la pila.
    - Si no, entonces los símbolos no están balanceados.

Por ejemplo, para la cadena `({[]})`, el procedimiento sería el siguiente:

1. Se encuentra el símbolo de apertura `(`, por lo que se apila.
2. Se encuentra el símbolo de apertura `{`, por lo que se apila.
3. Se encuentra el símbolo de apertura `[`, por lo que se apila.
4. Se encuentra el símbolo de cierre `]`, por lo que se desapila `[`.
5. Se encuentra el símbolo de cierre `}`, por lo que se desapila `{`.
6. Se encuentra el símbolo de cierre `)`, por lo que se desapila `(`.

Finalmente, el endpoint debe devolver una respuesta JSON que indique si los símbolos están balanceados o no. Por ejemplo, para la cadena `({[]})`:

```
{
      "balanced": true
}
```

> La estructura de datos que mencionamos ha sido implementada en la materia de Algorítmica.