

Gestión de la Información en la Web  
Curso 2024–25  
Práctica 8 – Persistencia con MongoEngine

**Fecha de entrega: domingo 24 de noviembre de 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 `pr8.py`. El esqueleto de este fichero se puede descargar del Campus Virtual.

**Lenguaje de programación**

**Python 3.11** o superior.

**Evaluación**

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 deshonesto 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 definiremos y usaremos un esquema MongoEngine que nos permitirá tener persistencia de datos en el contexto de una tienda *online*. La persistencia se debe realizar sobre una base de datos llamada **giw\_\_mongoengine** y debe soportar los siguientes *conceptos*:

### Importante

El nombre de las clases y de los atributos de cada clase debe ser **exactamente el mismo** que aparece en el enunciado. En caso de duda, podéis buscar en el fichero `tests.py` para ver más ejemplos de creación de objetos.

### Usuario

- **dni**: DNI con letra del usuario (obligatorio, único, cumpliendo el formato español)
- **nombre**: Nombre del usuario (obligatorio, mínimo 2 letras)
- **apellido1**: Primer apellido (obligatorio, mínimo 2 letras)
- **apellido2**: Segundo apellido (opcional)
- **f\_nac**: Fecha de nacimiento (obligatorio, formato de cada entrada AAAA-MM-DD)
- **tarjetas**: Lista de **tarjetas de crédito** anidadas (opcional)
- **pedidos**: Lista de referencias a **pedidos** (opcional)

### Ejemplo:

```
Usuario(dni="65068806N", nombre="Pepe", apellido1="Peces", apellido2="Cuadrado",  
        f_nac="1985-12-11", tarjetas=..., pedidos=...)
```

### Tarjeta

- **nombre**: Nombre del propietario (obligatorio, mínimo 2 letras)
- **numero**: Número de la tarjeta (obligatorio, 16 dígitos)
- **mes**: Mes de caducidad (obligatorio, 2 dígitos)
- **year**<sup>1</sup>: Año de caducidad (obligatorio, 2 dígitos)
- **cvv**: Valor de verificación de tarjeta (obligatorio, 3 dígitos)

### Ejemplo:

```
Tarjeta(nombre="Pepe", numero="4500874512587896", mes="12", year="19", cvv="852")
```

---

<sup>1</sup>Este nombre está en inglés para evitar incluir identificadores Python con caracteres no ASCII, como sería *año*.

## Pedido

- **total**: Precio total del pedido (número positivo, obligatorio)
- **fecha**: Fecha del pedido (obligatorio, formato AAAA,MM,DD,HH,MM,SS,NNNNNN)
- **lineas**: Lista de **líneas del pedido** anidadas (obligatorio)

### Ejemplo:

```
Pedido(total=4.5, fecha="2016,11,25,10,15,24,000000", lineas=...)
```

## Línea (Línea de pedido)

- **num\_items**: Cantidad de productos comprados (obligatorio, número natural positivo)
- **precio\_item**: Precio de un producto (obligatorio, número real positivo)
- **nombre\_item**: Nombre del producto (obligatorio, mínimo 2 letras)
- **total**: Precio total de la línea (obligatorio, número real positivo)
- **producto**: Referencia al **producto** (obligatorio)

### Ejemplo:

```
Línea(num_items=3, precio_item=1.5, nombre_item="Galletas Oreo", total=4.5, ref=...)
```

## Producto

- **codigo\_barras**: Código de barras (obligatorio, único, cumpliendo el formato EAN-13)
- **nombre**: Nombre (obligatorio, mínimo 2 letras)
- **categoria\_principal**: Categoría principal (obligatorio, número natural)
- **categorias\_secundarias**: Lista de categorías secundarias (opcional, números naturales)

### Ejemplo:

```
Producto(codigo_barras="9780201379624", nombre="Galletas Oreo", categoria_principal=5,  
categorias_secundarias=[5, 8, 9])
```

## 1. Definición del esquema

Crea en un fichero `pr8` las clases necesarias para representar en MongoEngine el esquema anteriormente detallado. Además de las comprobaciones naturales (longitudes, valores mínimos, etc.) el esquema debe realizar las siguientes validaciones adicionales:

1. El formato del DNI de los usuarios es correcto, incluyendo la letra de control: <https://www.interior.gob.es/opencms/ca/servicios-al-ciudadano/tramites-y-gestiones/dni/calculo-del-digito-de-control-del-nif-nie/>.
2. El precio total de un pedido es exactamente la suma de los precios de todas sus líneas.

3. El precio total de una línea de pedido está correctamente calculado en base al precio de un producto y la cantidad de productos comprados.
4. El nombre del producto de una línea de pedido es el mismo que el del producto al que apunta su referencia.
5. En un pedido, no puede haber dos líneas diferentes asociadas a un mismo producto.
6. El código de barras EAN-13 de un producto está bien formado, incluyendo su dígito de control: [https://en.wikipedia.org/wiki/International\\_Article\\_Number](https://en.wikipedia.org/wiki/International_Article_Number).
7. Si un producto tiene categorías secundarias, su categoría principal debe aparecer en el primer lugar de esa lista.
8. Cuando un pedido se elimina, este debe desaparecer de la lista de pedidos del usuario que lo realizó.

## 2. Tests de unidad

Implementar correctamente el esquema de MongoEngine teniendo en cuenta todas las restricciones del enunciado requiere bastante cuidado con los detalles. Para que podáis comprobar con facilidad que vuestro esquema cumple con los requerimientos, tenéis una serie de tests de unidad que crean objetos usando valores correctos e incorrectos. 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 tenéis que colocar el fichero `tests.py` en la misma carpeta donde tengáis vuestra práctica `pr8.py`. Luego, desde un terminal en esa misma carpeta debéis escribir lo siguiente:

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

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

- Para lanzar **un test concreto**, por ejemplo `test_usuarios_validos`:

```
$ python -m unittest tests.TestPersistencia.test_usuarios_validos
```

**Importante:** para poder ejecutar los tests de unidad, debéis tener un servidor MongoDB ejecutándose en la máquina local y escuchando en el puerto por defecto.