Validaciones Zod/Zon

Afondamento nas Competencias Profesionais

Cristian Fernández

Sábado, 21 de febrero de 2026

Índice

1. Introducción

El objetivo de esta práctica es aplicar la librería **Zon** en un proyecto en construcción que tiene como objetivo gestionar las actividades y reservas de un gimnasio. Dicho proyecto está configurado de la siguiente manera:

- **Backend API.** El backend está programado en python usando el framework flask con base de datos en MongoDB. Fue mi elección por 2 razones, la primera porque nunca había "programado" en python y la segunda porque flask tiene fama de ser muy agradable de programar.
- **Frontend Escritorio.** El frontend de escritorio está programado en Vue y Electron y la razón es porque las especificaciones lo obligaban.
- **Frontend Móvil.** El frontend móvil está programado en React Native. Las opciones eran ésta o Flutter y me decidí por React por estar ya familiarizado por una práctica anterior y porque es muy sencillo compilarlo en máquina real.

2. ¿Por qué implementarlo?

La razón principal para implementar esta librería es sencilla: **SIMPLIFICACIÓN**.

Zon nos permitirá definir esquemas declarativos y legibles y validar datos en runtime. Esto nos proporcionará ventajas tales como un código menos repetitivo y más simple, tener las reglas centralizadas en un solo esquema, mensajes de error más consistentes, etc.

La librería se implementará en el Backend, y específicamente es la siguiente:

- *zon (pip install zon)*

Se puede encontrar en https://pypi.org/

3. ¿Qué es Zod/Zon?

Zod/Zon es una biblioteca de validación de esquemas, diseñada para garantizar la integridad y el tipo de los datos en tiempo de ejecución de forma sencilla y expresiva. Permite definir estructuras de datos (strings, números, objetos, arrays) una única vez.

Características principales:

- **Seguridad de tipos.** Ofrece una inferencia de tipos automática y precisa, asegurando que los datos cumplan con las reglas definidas.
- **Validación en tiempo de ejecución.** Verifica que los datos recibidos (por ejemplo, desde una API o un formulario) cumplan con el esquema.
- **Sintaxis declarativa.** Se definen los datos mediante esquemas, por ejemplo, z.string().email(), lo que facilita la lectura y el mantenimiento.
- **Mensajes de error personalizables.** Permite generar errores claros y personalizados si los datos no coinciden con el esquema.

4. Implementación

Crearemos un nuevo archivo en nuestro backend en el que alojar los esquemas de validación para el registro de usuarios y el login.

Así quedaría el código.

```python
import zon

# Esquema para registro de usuario
validacionRegistro = zon.record({
    "nombre_usuario": zon.string().min(4).max(10),
    "contraseña": zon.string().min(8),
    "nombre": zon.string().max(12),
    "apellidos": zon.string().max(22),
    "email": zon.string().email(),
    "dni": zon.string().regex(r'^[0-9]{8}[A-Z]$'),
    "telefono": zon.string().regex(r'^[0-9]{9}$').optional()
})

# Esquema para login de usuario
validacionLogin = zon.record({
    "nombre_usuario": zon.string().min(4).max(10),
    "contraseña": zon.string().min(8)
})
```

Figura 1: Archivo *validaciones.py*, los esquemas son muy legibles

Debemos importar los esquemas en nuestra API.

```python
from flask_bcrypt import Bcrypt
from flask_jwt_extended import JWTManager, create_access_token
from flask_cors import CORS
from validaciones import validacionRegistro, validacionLogin

```

Echemos un vistazo al comienzo del endpoint de registro.

```python
## REGISTRAR USUARIO EN APP
@app.route('/auth/registro', methods=['POST'])
def registroUsuario():
    coleccion = db['usuarios']
    datos = request.json

    # VALIDACIONES
    try:
        # Intentamos validar los datos obtenidos del registro
        datos = validacionRegistro.validate(datos)
    except Exception as e:
        # Si hay un error se especificará de forma detallada
        return jsonify({"ERROR": "Validación fallida", "detalle": str(e)}), 400

```

Ahora observemos el comienzo del endpoint de login.

```python
## LOGIN USUARIO EN APP
@app.route('/auth/login', methods=['POST'])
def loginUsuario():
    coleccion = db['usuarios']
    datos = request.json

    # VALIDACIONES
    try:
        # Intentamos validar los datos obtenidos del login
        datos = validacionLogin.validate(datos)
    except Exception as e:
        # Si hay un error se especificará de forma detallada
        return jsonify({"ERROR": "Validación fallida", "detalle": str(e)}), 400

```

En el comienzo de ambos endpoints se puede observar algo muy interesante y es la tremenda simplificación de código, ya que todo la parte de validación se reduce a 4 líneas.

Veamos en la siguiente página el proceso de registrar un nuevo usuario dentro del frontend de escritorio.

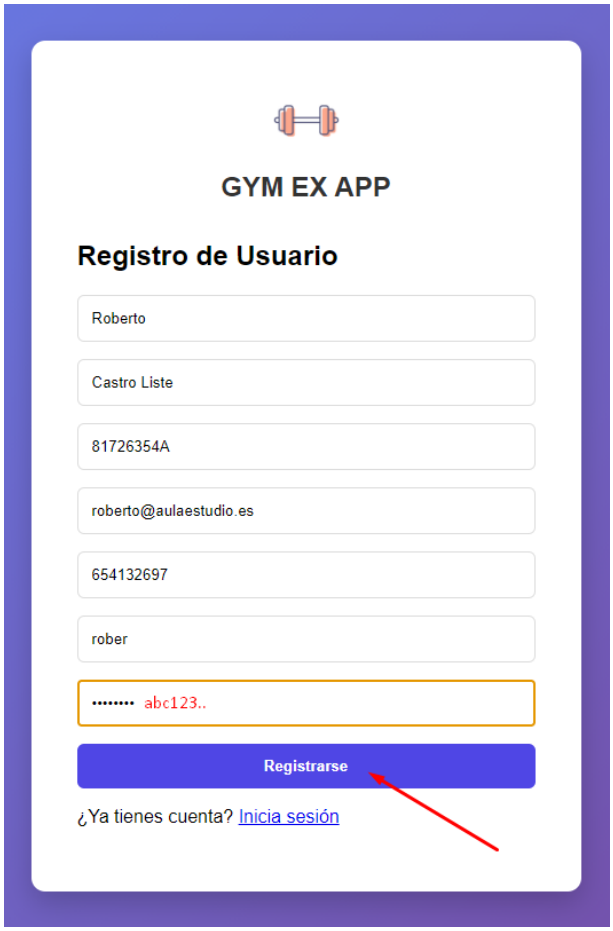

Figura 2: Vista en desarrollo

Comprobamos el nuevo registro en la base de datos


```
_id: ObjectId('69990fd94a0f8b020673074f')
nombre_usuario : "rober"
contraseña : "$2b$12$sfK1jUoXB3.tzdMWMfTHZOsFYjIZ0q4xOlRsTTvkaYcF3WjhmHId6"
nombre : "Roberto"
apellidos : "Castro Liste"
dni : "81726354A"
telefono : "654132697"
email : "roberto@aulaestudio.es"
rol : "cliente"
fecha_alta : 2026-02-21T02:52:25.217+00:00
estado_suscripcion : true
```


Como se puede apreciar, la validación del formulario ha sido satisfactoria.