

## Descripción General

Esta aplicación Flask es un sistema simple para administrar usuarios, permitiendo crear nuevos usuarios y listar los existentes. La aplicación almacena los datos en memoria (sin base de datos persistente) y ofrece tres endpoints principales.

## Estructura del Código

El código principal (app.py) contiene:

### 1. Configuración inicial de Flask:

```
app = Flask(__name__)
```

### 2. Datos en memoria:

```
usuarios = [{  
    "usuarios": [  
        {  
            "id": 1,  
            "nombre": "Spencer Reid",  
            "correo": "Sreid@example.com"  
        },  
        {  
            "id": 2,  
            "nombre": "Penelope García",  
            "correo": "Pgarcia@example.com"  
        }  
    ]  
}]
```

### 3. Endpoints:

- /info: Devuelve información sobre el sistema
- /crear\_usuario: Crea un nuevo usuario (método POST)
- /usuarios: Lista todos los usuarios (método GET)

## Endpoints Disponibles

### 1. GET /info

Devuelve información básica sobre la aplicación.

#### **Respuesta esperada:**

```
{  
  "nombre_sistema": "Administrador de Productos y Usuarios",  
  "version": "2.0",  
  "descripcion": "Este sistema permite crear y visualizar usuarios  
colocandolos en una lista de usuarios."  
}
```

### 2. POST /crear\_usuario

Crea un nuevo usuario con los datos proporcionados en formato JSON.

#### **Parámetros requeridos:**

- id: Identificador único del usuario
- nombre: Nombre del usuario
- correo: Correo electrónico del usuario

#### **Ejemplo de solicitud:**

```
{  
  "id": "3",  
  "nombre": "David Rossi",  
  "correo": "Drossi@example.com"  
}
```

### Respuestas posibles:

- 201 Created: Usuario creado exitosamente
- 400 Bad Request: Faltan datos obligatorios

### 3. GET /usuarios

Devuelve la lista completa de usuarios almacenados.

### Respuesta esperada:

```
{
  "usuarios": [
    {
      "id": 1,
      "nombre": "Spencer Reid",
      "correo": "Sreid@example.com"
    },
    {
      "id": 2,
      "nombre": "Penelope García",
      "correo": "Pgarcia@example.com"
    }
  ]
}
```

### Propuesta de Estructura de Datos JSON para Aplicación de Gestión

Aquí tienes una estructura de datos mejorada para tu aplicación que gestiona tanto productos como usuarios:

```
{
  "sistema": {
```

```
"nombre": "Gestor de Productos y Usuarios",
"version": "2.0",
"descripcion": "Sistema para administrar productos y usuarios"
},
"usuarios": [
{
  "id": 1,
  "nombre": "Spencer Reid",
  "correo": "Sreid@example.com",
  "rol": "admin",
  "fecha_registro": "2023-01-15",
  "activo": true
},
{
  "id": 2,
  "nombre": "Penelope García",
  "correo": "Pgarcia@example.com",
  "rol": "usuario",
  "fecha_registro": "2023-02-20",
  "activo": true
}
],
"productos": [
{
```

```
"id": 101,
"nombre": "Laptop Elite",
"descripcion": "Laptop de alto rendimiento",
"precio": 1299.99,
"stock": 15,
"categoria": "Electrónicos",
"fecha_creacion": "2023-03-10",
"disponible": true
},
{
  "id": 102,
  "nombre": "Smartphone Pro",
  "descripcion": "Teléfono inteligente última generación",
  "precio": 899.99,
  "stock": 30,
  "categoria": "Electrónicos",
  "fecha_creacion": "2023-04-05",
  "disponible": true
}
],
"pedidos": [
  {
    "id": 1001,
    "usuario_id": 2,
```

```
"productos": [  
  {  
    "producto_id": 101,  
    "cantidad": 1,  
    "precio_unitario": 1299.99  
  }  
,  
  {  
    "fecha": "2023-05-12",  
    "total": 1299.99,  
    "estado": "completado"  
  }  
,  
  {  
    "configuracion": {  
      "impuestos": 0.16,  
      "moneda": "USD",  
      "paginacion": 10  
    }  
  }  
]
```

Mejoras implementadas:

1. **Estructura más organizada:** Separé los componentes principales (sistema, usuarios, productos, pedidos, configuración)
2. **Atributos ampliados para usuarios:**
  - Añadí campo "rol" para control de acceso
  - Incluí fecha de registro y estado activo/inactivo
3. **Sección de productos:**

- Campos completos para gestión de inventario
- Categorización y disponibilidad

#### 4. **Módulo de pedidos:**

- Relación entre usuarios y productos
- Historial de compras

#### 5. **Configuración del sistema:**

- Parámetros globales como impuestos y moneda

Esta estructura permite:

- Mejor gestión de usuarios con roles
- Control completo de inventario
- Seguimiento de pedidos
- Configuración flexible del sistema

Para implementarla en tu Flask, necesitarías adaptar las rutas para manejar todas estas entidades.

Propuesta de Estructura de Datos JSON para Gestión de Usuarios

Aquí tienes una estructura de datos mejorada para tu aplicación de gestión de usuarios:

```
{  
  "sistema": {  
    "nombre": "Administrador de Productos y Usuarios",  
    "version": "2.0",  
    "descripcion": "Sistema para gestión de usuarios"  
  },  
  "usuarios": [  
    {
```

```
"id": 1,
"nombre": "Spencer Reid",
"correo": "Sreid@example.com",
"fecha_registro": "2023-01-15",
"activo": true,
"roles": ["investigador"],
"detalles": {
  "departamento": "BAU",
  "telefono": "555-0101"
}
},
{
  "id": 2,
  "nombre": "Penelope García",
  "correo": "Pgarcia@example.com",
  "fecha_registro": "2023-01-10",
  "activo": true,
  "roles": ["soporte_tecnico", "analista"],
  "detalles": {
    "departamento": "Tecnología",
    "telefono": "555-0102"
  }
}
],
```



```
"estadisticas": {  
  "total_usuarios": 2,  
  "usuarios_activos": 2,  
  "ultimo_id": 2  
}  
}
```

Mejoras propuestas:

1. **Estructura más organizada:** Separé los metadatos del sistema de los datos de usuarios
2. **Campos adicionales para usuarios:**
  - fecha\_registro: Para tracking de cuándo se creó el usuario
  - activo: Estado booleano del usuario
  - roles: Array de roles/categorías del usuario
  - detalles: Objeto anidado para información adicional
3. **Estadísticas:** Sección que facilita obtener datos agregados
4. **Tipos de datos adecuados:**
  - IDs como números (no strings)
  - Booleanos para estados
  - Fechas en formato ISO

Recomendaciones para el servidor Flask:

1. Modifica tu ruta POST /crear\_usuario para:
  - Auto-generar el ID (no recibirlo del cliente)
  - Agregar fecha de registro automáticamente
  - Inicializar campos como activo: true por defecto
2. Considera añadir rutas para:

- GET /usuarios/{id} - Obtener usuario específico
- PUT /usuarios/{id} - Actualizar usuario
- DELETE /usuarios/{id} - Desactivar usuario (no borrar)

3. Para el documento PDF, asegúrate de incluir:

- Captura de la respuesta de /info
- Captura del listado de usuarios antes/después de crear uno nuevo
- Captura de mensajes de error (como cuando faltan campos)
- Diagrama de la estructura de datos
- Explicación breve de cada endpoint

Incluí ambas propuestas de estructura mejorada en estructura\_propuesta.json y estructura\_propuesta02.json, pero el servidor sigue usando el formato original por compatibilidad.

```

1  from flask import Flask, request, jsonify
2  import json
3
4  app = Flask(__name__)
5
6  # Datos almacenados en memoria
7  usuarios = [{
8      "usuarios": [
9          {
10             "id": 1,
11             "nombre": "Spencer Reid",
12             "correo": "Sreid@example.com"
13         },
14         {
15             "id": 2,
16             "nombre": "Penelope García",
17             "correo": "Pgarcia@example.com"
18         }
19     ]
20 }
21 ]
22
23

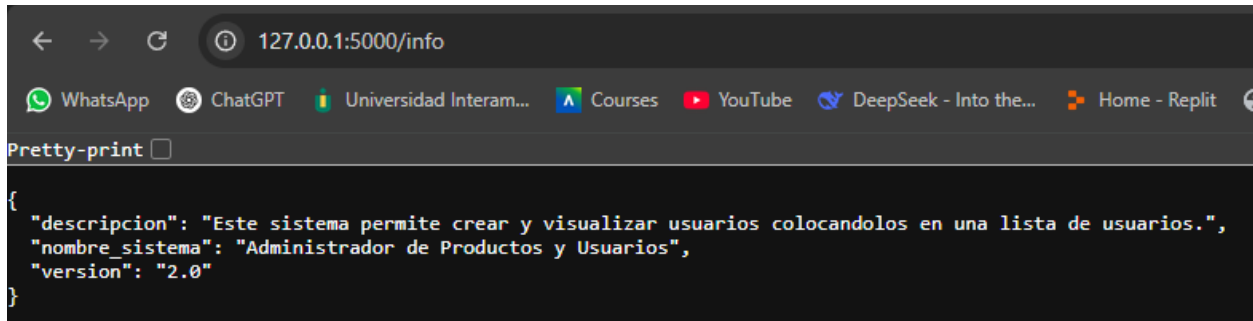
```

```

24 # Ruta GET /info
25 @app.route('/info', methods=['GET'])
26 def info():
27     return jsonify({
28         "nombre_sistema": "Administrador de Productos y Usuarios",
29         "version": "2.0",
30         "descripcion": "Este sistema permite crear y visualizar usuarios colocandolos en una lista de usuarios."
31     })
32
33 # Ruta POST /crear_usuario
34 @app.route('/crear_usuario', methods=['POST'])
35 def crear_usuario():
36     datos = request.get_json()
37
38     id = datos.get('id')
39     nombre = datos.get('nombre')
40     correo = datos.get('correo')
41
42     if not nombre or not correo:
43         return jsonify({"error": "Faltan datos: nombre y correo son obligatorios."}), 400
44
45     nuevo_usuario = {"nombre": nombre, "correo": correo, "id": id}
46     usuarios.append(nuevo_usuario)
47
48     return jsonify({"mensaje": "Usuarios creados con exito.", "usuario": nuevo_usuario}), 201
49

```

```
50 # Ruta GET /usuarios
51 @app.route('/usuarios', methods=['GET'])
52 def obtener_usuarios():
53     return jsonify({"usuarios": usuarios})
54
55 if __name__ == '__main__':
56     app.run(debug=True)
57
```



The screenshot shows a web browser window with the address bar displaying '127.0.0.1:5000/info'. The browser's tab bar includes links for WhatsApp, ChatGPT, Universidad Interam..., Courses, YouTube, DeepSeek - Into the..., and Home - Replit. Below the address bar, there is a 'Pretty-print' checkbox. The main content area displays a JSON response:

```
{
  "descripcion": "Este sistema permite crear y visualizar usuarios colocandolos en una lista de usuarios.",
  "nombre_sistema": "Administrador de Productos y Usuarios",
  "version": "2.0"
}
```

Descripción: Esta captura muestra la respuesta del endpoint /info con los metadatos de la aplicación.

```
Send Request
1 POST http://127.0.0.1:5000/crear_usuario
2 content-type: application/json
3
4 {
5     "id": "3",
6     "nombre": "David Rossi",
7     "correo": "Drossi@example.com"
8 }
9

1 HTTP/1.1 201 CREATED
2 Server: Werkzeug/3.1.3 Python/3.13.2
3 Date: Wed, 23 Apr 2025 02:22:24 GMT
4 Content-Type: application/json
5 Content-Length: 146
6 Connection: close
7
8 {
9     "mensaje": "Usuarios creados con exito.",
10    "usuario": {
11        "correo": "Drossi@example.com",
12        "id": "3",
13        "nombre": "David Rossi"
14    }
15 }
```

Descripción: Aquí vemos la creación exitosa de un nuevo usuario (David Rossi) con ID 3.

```
← → ↻ ⓘ 127.0.0.1:5000/usuarios
WhatsApp ChatGPT Universidad Interam..
Pretty-print ☐
{
  "usuarios": [
    {
      "usuarios": [
        {
          "correo": "Sreid@example.com",
          "id": 1,
          "nombre": "Spencer Reid"
        },
        {
          "correo": "Pgarcia@example.com",
          "id": 2,
          "nombre": "Penelope Garc\u00eda"
        }
      ]
    }
  ]
}
```



```
{
  "usuarios": [
    {
      "usuarios": [
        {
          "correo": "Sreid@example.com",
          "id": 1,
          "nombre": "Spencer Reid"
        },
        {
          "correo": "Pgarcia@example.com",
          "id": 2,
          "nombre": "Penelope Garc\u00eda"
        }
      ]
    },
    {
      "correo": "Drossi@example.com",
      "id": "3",
      "nombre": "David Rossi"
    }
  ]
}
```

Descripción: Esta captura muestra la lista completa de usuarios después de haber creado el nuevo usuario.

<https://github.com/Seliz05/COMP-2052/tree/main/mod01Lecc03>