



Instituto Politécnico Nacional.  
Unidad Profesional Interdisciplinaria en  
Ingeniería y Tecnologías Avanzadas.



## **Ingeniería Telemática.**

Aplicaciones Distribuidas

### **Práctica de repaso**

Bautista Uribe Juan Manuel.

Martínez Aparicio Eduardo

**Profesor: Sierra Romero Noe**

**4TM3.**

# Saludo

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:3000/saludo`. The request method is `POST`. The response status is `200 OK` with a response time of `14 ms` and a body size of `277 B`. The response content is `Hola, Eduardo, lupita y juan`.

# Calculadora

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:3000/calcular`. The request method is `POST`. The response status is `200 OK` with a response time of `15 ms` and a body size of `251 B`. The response content is `13`.

The image displays three separate Postman requests to the endpoint `http://localhost:3000/calcular`. Each request is a POST method with the following body:

```

1  {
2    "a": 10,
3    "b": 2,
4    "operacion": "division"
5  }
6

```

**Request 1 (Successful):**

Body: `{}` JSON | Preview | Visualize | `resultado`: 5

**Request 2 (Error: Division by zero):**

Body: `{}` JSON | Preview | Debug with AI | `error`: No se puede dividir entre cero.

**Request 3 (Error: Invalid operation):**

Body: `{}` JSON | Preview | Debug with AI | `error`: Operación no válida. Use: suma, resta, multiplicacion o division.

## Gestor de tareas

The image shows a single Postman request to the endpoint `http://localhost:3000/tareas`. The request method is POST with the following body:

```

1  {
2    "id": 1,
3    "titulo": "Estudiar para el examen",
4    "completada": false
5  }
6

```

**Response:**

201 Created | 47 ms | 354 B | `mensaje`: Tarea creada exitosamente

tareas [1]		
	id	título
	0	1
		Estudiar para el examen
		completada
		false

The image displays two side-by-side Postman requests for validating a password.

**Request 1 (Left):**

- Method: POST
- URL: http://localhost:3000/validar-password
- Body (raw JSON):

```
{
  "password": "clave"
}
```

**Request 2 (Right):**

  - Method: POST
  - URL: http://localhost:3000/validar-password
  - Body (raw JSON):

```
{
  "password": "ClaveSegura"
}
```

Both requests show a response with status 200 OK, indicating validation errors:

  - Request 1 Response:** esValida: false, errores: [3]
    - 0: La contraseña debe tener al menos 8 caracteres.
    - 1: Debe contener al menos una letra mayúscula.
    - 2: Debe contener al menos un número.
  - Request 2 Response:** esValida: false, errores: [1]
    - 0: Debe contener al menos un número.

## Validador de contraseñas

The image shows a Postman request for converting temperature from Celsius to Fahrenheit.

**Request:**

- Method: POST
- URL: http://localhost:3000/convertir-temperatura
- Body (raw JSON):

```
{
  "valor": 25,
  "desde": "C",
  "hacia": "F"
}
```

**Response:**

- Status: 200 OK
- Time: 68 ms
- Size: 320 B
- Body (JSON):

```

{
  "valorOriginal": 25,
  "valorConvertido": 77,
  "escalaOriginal": "C",
  "escalaConvertida": "F"
}

```

# Convertidor temperatura

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost:3000/validar-password`. The request method is POST, and the body contains the JSON `{"password": "claveSegura1"}`. The response status is 200 OK, with a response time of 64 ms and a size of 285 B. The response body is `{ "esValida": true, "errores": [] }`.

The screenshot shows three parallel API calls in Postman. All three calls have the same URL: `http://localhost:3000/convertir-temperatura` and the same POST method. The first two calls succeed, while the third one fails.

- Call 1:** Body: `{"valor": 32, "desde": "F", "hacia": "K"}`. Response: `valorOriginal: 32, valorConvertido: 273.15, escalaOriginal: F, escalaConvertida: K`.
- Call 2:** Body: `{"valor": 300, "desde": "K", "hacia": "C"}`. Response: `valorOriginal: 300, valorConvertido: 26.85, escalaOriginal: K, escalaConvertida: C`.
- Call 3:** Body: `{"valor": 25, "desde": "X", "hacia": "F"}`. Response: `error: Datos inválidos. Use escalas: C, F o K.`

# Buscador array

The first request (left) has the JSON body:

```
1 {  
2   "array": ["manzana", "pera", "uva"],  
3   "elemento": "naranja"  
4 }  
5
```

The second request (middle) has the JSON body:

```
1 {  
2   "array": [true, false, true],  
3   "elemento": false  
4 }  
5
```

The third request (right) has the JSON body:

```
1 {  
2   "array": "no soy un arreglo",  
3   "elemento": 5  
4 }  
5
```

Below each request, the response preview shows the results:

- encontrado: false
- indice: -1
- tipoElemento: string

- encontrado: true
- indice: 1
- tipoElemento: boolean

- encontrado: true
- indice: 1
- tipoElemento: boolean
- error: El campo "array" debe ser un arreglo válido.

The request body is:

```
1 {  
2   "array": [10, 20, 30, 40, 50],  
3   "elemento": 30  
4 }  
5
```

The response preview shows the results:

- encontrado: true
- indice: 2
- tipoElemento: number

# Contador de palabras

The screenshot shows the Postman interface with a successful API call to `http://localhost:3000/contar-palabras`. The request method is POST, and the body is a JSON object with a single key `texto` containing the value `"Hola mundo, esto es un ejemplo de texto."`. The response status is 200 OK, and the results show the following metrics:

Metrica	Valor
totalPalabras	8
totalCaracteres	40
palabrasUnicas	8

The screenshot shows two failed API calls to `http://localhost:3000/contar-palabras`. Both requests are POST methods with invalid JSON bodies. The first request has a body with `texto` containing multiple spaces: `"Hola hola mundo mundo"`. The second request has a body with `texto` containing a number: `12345`. Both requests result in an error message: `Debe enviar un texto válido.`

# Generador de perfil

The image displays three side-by-side Postman requests for generating user profiles. Each request is a POST to `http://localhost:3000/generar-perfil`.

- Request 1 (Left):** Body is raw JSON:

```
1 {
2   "nombre": "Eduardo",
3   "edad": 38,
4   "intereses": ["programación", "lectura", "viajes"]
5 }
```

- Request 2 (Middle):** Body is raw JSON:

```
1 {
2   "nombre": "juan",
3   "edad": 38,
4   "intereses": ["programación", "lectura", "viajes"]
5 }
```

- Request 3 (Right):** Body is raw JSON:

```
1 {
2   "nombre": "lupita",
3   "edad": 38,
4   "intereses": ["programación", "lectura", "viajes"]
5 }
```

Each request shows a successful response with status 200 OK, 66 ms duration, and 318 B size.

# Sistema de calificaciones

The image shows a Postman request for calculating a grade average.

**Request:** POST `http://localhost:3000/calcular-promedio`

**Body:** raw JSON

```
1 {
2   "calificaciones": [8, 9, 7, 6, 10]
3 }
```

**Response:** 200 OK

Test results show the following data:

Variable	Value
promedio	8
calificacionMasAlta	10
calificacionMasBaja	6
estado	aprobado

The image shows two side-by-side Postman requests to the endpoint `http://localhost:3000/calcular-promedio`.

**Request 1 (Left):**

- Method: POST
- URL: `http://localhost:3000/calcular-promedio`
- Body (raw JSON):

```

1 {
2   "calificaciones": [4, 5, 3, 6]
3 }
4

```

**Response 1 (Left):**

promedio	4.5
calificacionMasAlta	6
calificacionMasBaja	3
estado	reprobado

**Request 2 (Right):**

- Method: POST
- URL: `http://localhost:3000/calcular-promedio`
- Body (raw JSON):

```

1 {
2   "calificaciones": [8, 11, 5]
3 }
4

```

**Response 2 (Right):**

error: Todas las calificaciones deben ser números entre 0 y 10.

## Api-products

The image shows a Postman request to the endpoint `http://localhost:3000/productos`.

**Request:**

- Method: POST
- URL: `http://localhost:3000/productos`
- Body (raw JSON):

```

1 {
2   "id": 1,
3   "nombre": "Laptop",
4   "categoria": "Electrónica",
5   "precio": 1500
6 }
7

```

**Response:**

201 Created

mensaje: Producto agregado

productos [4]

id	1
nombre	Laptop
categoria	Electrónica
precio	1500

POST http://localhost:3000/productos

HTTP **http://localhost:3000/productos**

POST http://localhost:3000/productos

Params Authorization Headers (8) Body

none  form-data  x-www-form-urlencoded

```
1  {
2    "id": 2,
3    "nombre": "Cafetera",
4    "categoria": "Hogar",
5    "precio": 200
6  }
```

Body Cookies Headers (7) Test Results

{ } JSON  Preview  Visualize

Root / producto

<b>id</b>	2
<b>nombre</b>	Cafetera
<b>categoria</b>	Hogar
<b>precio</b>	200

POST http://localhost:3000/productos

HTTP **http://localhost:3000/productos**

POST http://localhost:3000/productos

POST http://localhost:3000/productos

POST http://localhost:3000/productos

POST http://localhost:3000/productos

Params Authorization Headers (8) Body Scripts Tests Settings

none  form-data  x-www-form-urlencoded  raw  binary  GraphQL J

```
1  {
2    "id": "x",
3    "nombre": 123,
4    "categoria": "Electrónica",
5    "precio": "1500"
6  }
```

Body Cookies Headers (7) Test Results

{ } JSON  Preview  Debug with AI

error Datos inválidos. Se requiere { id:number, nombre:string, categoria:string, precio:number }