
Programación 2

— API REST —

Representational State Transfer es un modelo de arquitectura de software orientado a construir servicios web escalables conforme a una serie de buenas prácticas.

Las principales son:

- Separación de responsabilidades en modelo **cliente y servidor**. (Separa la interfaz de usuario y la lógica del servidor)
- Modelo en base a **recursos** identificados en forma unívoca y homogénea.
- Operaciones atómicas **sin estado**.
- Conjunto de operaciones acotado: **CRUD**.

¿Qué es CRUD? ¿ABM?

CRUD es el acrónimo de Create (Crear), Read (Leer), Update (Actualizar) y Delete (Borrar). Este concepto se utiliza para describir las cuatro operaciones básicas que pueden realizarse en la mayoría de las bases de datos y sistemas de gestión de información.

La abreviatura ABM (Alta, Baja y Modificación) hace referencia a formularios que permite dar de alta, baja y modificar registros en una tabla de una base de datos.

Convenciones

Los recursos son las entidades (nombres) sobre las que operar.

Se identifican con URLs (Uniform Resource Locator). Ejs:

- `https://api.e-learning.com/{recurso}/{id}`
- `https://api.e-learning.com/users`
- `https://api.e-learning.com/users/41123`
- `https://api.e-learning.com/courses`
- `https://api.e-learning.com/courses/23`

Métodos

Los métodos son las operaciones (verbos) que podemos realizar sobre los recursos. Están acotadas por los métodos disponibles en HTTP:

- **POST**: Create (crear)
- **GET**: Read (leer, obtener)
- **PUT**: Update (actualizar)
- **DELETE**: Delete (eliminar)

La mayoría de los escenarios se pueden resolver con estas cuatro operaciones.

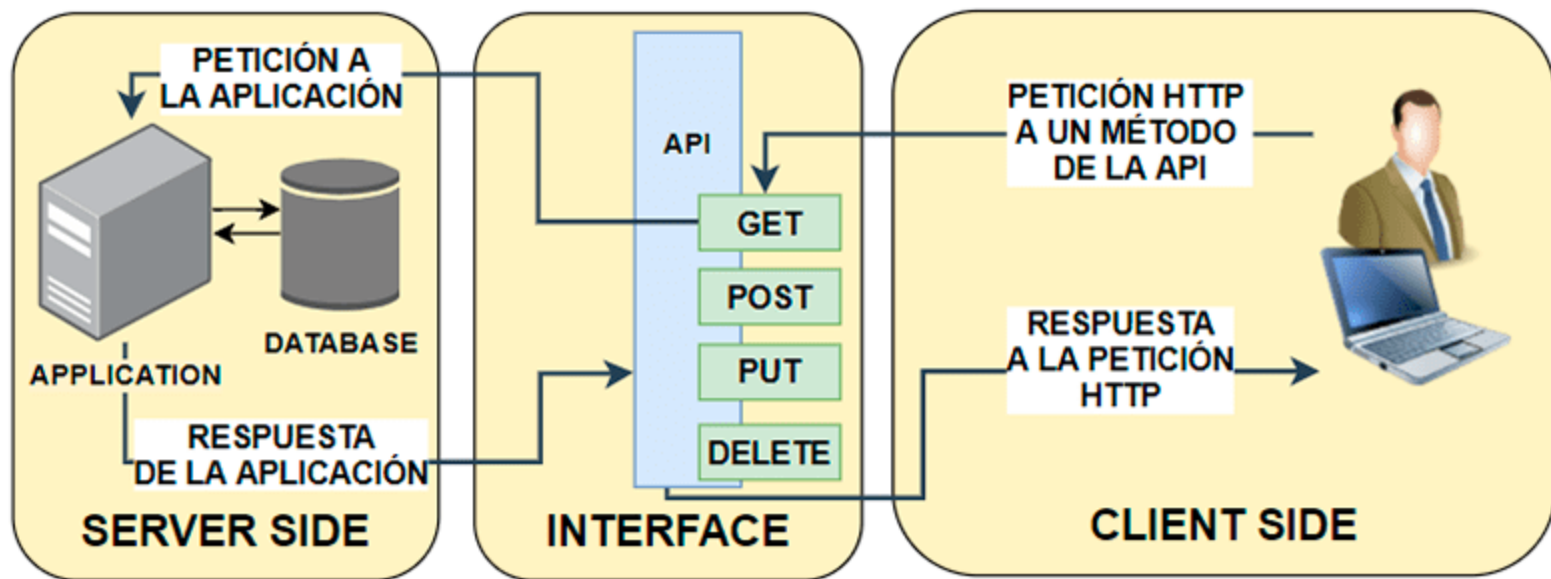
Códigos de respuesta comunes

- **200** OK
- **201** Created
- **400** Bad request (solicitud incorrecta)
- **401** Unauthorized (no autorizado)
- **403** Forbidden (prohibido)
- **404** Not found (no se ha encontrado)
- **405** Method not allowed (método no permitido)

Más info:

<https://developer.mozilla.org/es/docs/Web/HTTP/Status>

https://www.w3schools.com/tags/ref_httpmessages.asp



Ejemplo GET (leer)

GET <http://api.e-learning.com/courses>

200 OK

```
[
  {
    "id":1,
    "name":"Introducción a REST",
    "description":"Conceptos fundamentales de las APIs REST: recursos, métodos, status...",
    "duration":3,
    "modules":[ ...]
  },
  {
    "id":2,
    "name":"Python para desarrolladores Java",
    "description":"...",
    "duration":4,
    "modules":[ ...]
  }, ...
]
```


GET http://api.e-learning.com/courses/1

200 OK

```
{  
  "id":1,  
  "name":"Introducción a REST",  
  "description":"Conceptos fundamentales de las APIs REST",  
  "duration":3,  
  "modules":[...]  
}
```

Ejemplo GET con parámetros

Normalmente se pasan como query parameters y se utilizan para añadir criterios adicionales de **filtrado**, **ordenación**, **paginado**...

```
GET http://api.e-learning.com/courses?name=Spark&page_size=10&sort=+name
[
  {
    "id":1,
    "name":"Introducción a Spark",
    ...
  },
  {
    ...
  }
]
```

Ejemplo POST (crear)

```
POST http://api.e-learning.com/courses
```

```
{  
  "name": "Introducción a Spark",  
  "description": "Análisis de datos masivos en memoria con Spark",  
  "duration": 10,  
  "modules": [...]  
}
```

```
201 CREATED
```

```
Location: /courses/42113
```

Ejemplo PUT (modificar)

```
PUT http://api.e-learning.com/courses/42113
{
  "name": "Introducción a Spark" ,
  "description": "Análisis de datos masivos en memoria con Spark" ,
  "duration": 15,
  "Modules": [...]
}
```

200 OK

Ejemplo DELETE (borrar)

DELETE http://api.e-learning.com /**courses/42113**

204 NO CONTENT

ó

200 OK

```
{  
  "name": "Introducción a Spark",  
  "description": "Análisis de datos masivos en memoria con Spark",  
  "duration": 15,  
  "modules": [...]  
}
```

Convención de nombres

```
/listUsers?id=123  
/getUsers?id=123  
/findUsers?user_id=123  
/users/find/123  
/users/get_by_id/123  
/deleteUser
```



```
/users: GET y POST  
/users/{id}: GET, PUT y DELETE
```



¿Cómo lo vamos a implementar?



Flask

¿Qué es Flask?

Flask es un “micro” Framework escrito en Python y concebido para facilitar el desarrollo de Aplicaciones Web bajo el patrón MVC (Modelo - Vista - Controlador).

La palabra “micro” no significa que sirva para proyectos pequeños o que nos permita hacer páginas web pequeñas sino que al instalar Flask tenemos solamente las herramientas necesarias para crear una aplicación web funcional pero si se necesita en algún momento una nueva funcionalidad hay un conjunto muy grande extensiones (plugins) que se pueden instalar con Flask que le van dotando de funcionalidad.

Patrón MVC

El patrón MVC es una manera o una forma de trabajar que permite diferenciar y separar responsabilidades en tres capas:

- El modelo de datos (los datos que van a tener en la App que normalmente están guardados en BD) y la manipulación de éstos.
- La vista (página HTML o la representación de los datos).
- El controlador (donde se gestiona las peticiones de la app web).

Esta organización del código en tres capas separadas permite que el proyecto se mantenga modular, limpio y escalable.

Qué es un Framework?

Actualmente en el desarrollo moderno de aplicaciones web se utilizan distintos Frameworks que son herramientas que nos dan un esquema de trabajo y una serie de utilidades y funciones que nos facilita y nos abstrae de la construcción de páginas web dinámicas.

Ventajas de usar un Framework

- Proporciona una estructura del proyecto, es decir, todas las Apps que estén construidas con Flask van a tener los mismos elementos y los mismos ficheros.
- Facilita la colaboración.
- Es fácil encontrar bibliotecas adaptadas al Framework.

Entornos virtuales

Es recomendable utilizar entornos virtuales para los siguientes proyectos en los que tendremos que instalar librerías.

De esta forma cada proyecto tendrá su entorno virtual con las librerías que necesita y evitaremos conflictos entre los distintos proyectos por las distintas versiones de las librerías que pueden llegar a requerir.

Los entornos virtuales son grupos independientes de bibliotecas de Python, uno para cada proyecto. Los paquetes instalados para un proyecto no afectarán a otros proyectos ni a los paquetes del sistema operativo.

Python viene incluido con el módulo **venv** para crear entornos virtuales.

Para crear un entorno virtual en python nos posicionamos con la terminal en la carpeta donde desarrollaremos el proyecto y ejecutamos:

```
python -m venv <nombre_carpeta_entorno_virtual>
```

```
ej.: python -m venv .venv
```

Entornos virtuales - activación

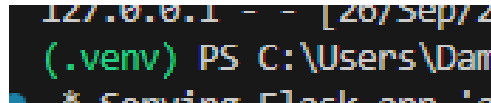
Si el comando anterior se ejecutó sin errores, sólo nos queda activar el entorno virtual ejecutando el script "activate":

```
cmd: .venv\Scripts\activate
```

Si no nos permite ejecutar scripts podemos modificar la política de ejecución de scripts en la carpeta indicada mediante el siguiente comando:

```
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
```

Una vez activado el entorno virtual, veremos la terminal del VSCode de la siguiente forma:



```
127.0.0.1 - - [26/Sep/2024:12:00:00]
(.venv) PS C:\Users\Damian>
* Copying Flask app to
```

Instalar Flask

Una vez creado y activado el entorno virtual para el desarrollo, procedemos a instalar Flask con el gestor de paquetes de python:

```
pip install Flask
```

Crear nuestro primer archivo **app.py**

ejecutar el siguiente comando para setear la variable de entorno y poder levantar el proyecto desde la consola:

```
set FLASK_APP=app.py
```

Añadir código a app.py

```
from flask import Flask

app = Flask(__name__) #creamos una instancia de la clase
Flask

@app.route('/')
def index():
    return '<h1>Hola!</h1>'
```

Alternativas para ejecutar:

- 1) Ejecutar el comando **flask run** (opcional modo debug: **--debug**)
- 2) Agregar **app.run** en el código condicionado para ejecutarse cuando se ejecuta el archivo como programa principal

1) comando flask run

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS

- Error: Could not locate a Flask application. Use the 'flask --app' option, 'FLASK_APP' environment variable, or a 'wsgi.py' or 'app.py' file in the current directory.

```
(miproxy) PS C:\Users\joel\proyflask\miproxy> cd ..
```

```
(miproxy) PS C:\Users\joel\proyflask> set FLASK_APP=app.py
```

```
(miproxy) PS C:\Users\joel\proyflask> flask run
```

- * Debug mode: off

WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.

* Running on http://127.0.0.1:5000

Press CTRL+C to quit

```
127.0.0.1 - - [02/Oct/2023 22:41:44] "GET / HTTP/1.1" 200 -
```

```
127.0.0.1 - - [02/Oct/2023 22:41:45] "GET /favicon.ico HTTP/1.1" 404 -
```

```
█
```


2) `app.run(debug=True)`

```
if __name__ == "__main__":  
    app.run(debug=True, port=4000)
```

`debug=True` nos proporcionará información detallada sobre errores, restablecerá el servidor cada vez que se detecte un cambio en el código, etc.

`port = 4000` nos permite cambiar el puerto por defecto del servidor, en este caso se va a ejecutar en `http://127.0.0.1:4000`

`__name__` es una variable especial de Python que representa el nombre del módulo actual.

Cuando un script es ejecutado, Python asigna el valor **`"__main__"`** a la variable `__name__` si es el script principal que está siendo ejecutado.

Si el script es importado como un módulo en otro script, entonces `__name__` de ese script tomará el valor del nombre del módulo (nombre del archivo).

Acceder a <http://127.0.0.1:5000/>

Disfrutar de su Hola!

Hola!

Código del ejemplo

```
from flask import Flask, request

app = Flask(__name__) #creamos una instancia de la clase Flask

@app.route("/signup/", methods=["GET"])
def metodo_que_procesa_peticion_get():
    return "No estas logeado. Esto es un get"

@app.route("/signup/", methods=[ "POST"])
def metodo_que_procesa_peticion_post():
    if request.is_json:
        data = request.get_json()
        name = data['name']
        email = data['email']
        password = data['password']
        return "Hola " + name + " " + email + " " + password
    return "No enviaste datos. Esto es un POST"

@app.route('/')
def index():
    return '<h1>Hola!</h1>'

if __name__ == '__main__':
    app.run(debug=True)
```

Postman!



POSTMAN

<https://www.postman.com/downloads/>

Postman en sus inicios nace como una extensión que podía ser utilizada en el navegador Chrome de Google y básicamente nos permite realizar peticiones de una manera simple para testear APIs de tipo REST propias o de terceros.

Para qué sirve Postman

Postman sirve para múltiples tareas dentro de las cuales destacaremos en esta oportunidad las siguientes:

- Testear colecciones o catálogos de APIs tanto para Frontend como para Backend.
- Organizar en carpetas, funcionalidades y módulos los servicios web.
- Permite gestionar el ciclo de vida (conceptualización y definición, desarrollo, monitoreo y mantenimiento) de nuestra API.
- Generar documentación de nuestras APIs.
- Trabajar con entornos (calidad, desarrollo, producción) y de este modo es posible compartir a través de un entorno cloud la información con el resto del equipo involucrado en el desarrollo.

Enviando una solicitud

- Hacer clic en **+** para abrir una nueva pestaña
- Seleccionar el método e Ingresar la URL
- Hacer clic en **send** para enviar la solicitud (request)

The screenshot displays the Postman API client interface. At the top, a new tab is created by clicking the '+' icon. The interface shows a GET request to `https://postman-echo.com/get`. The 'Send' button is highlighted. Below the request bar, the 'Query Params' section is visible, showing a table with 'Key' and 'Value' columns. The 'Body' tab is selected, showing the response in 'Pretty' format. The response is a JSON object with various headers and a 'url' field.

GET Postman API No Environment

Postman API / Postman API Save Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

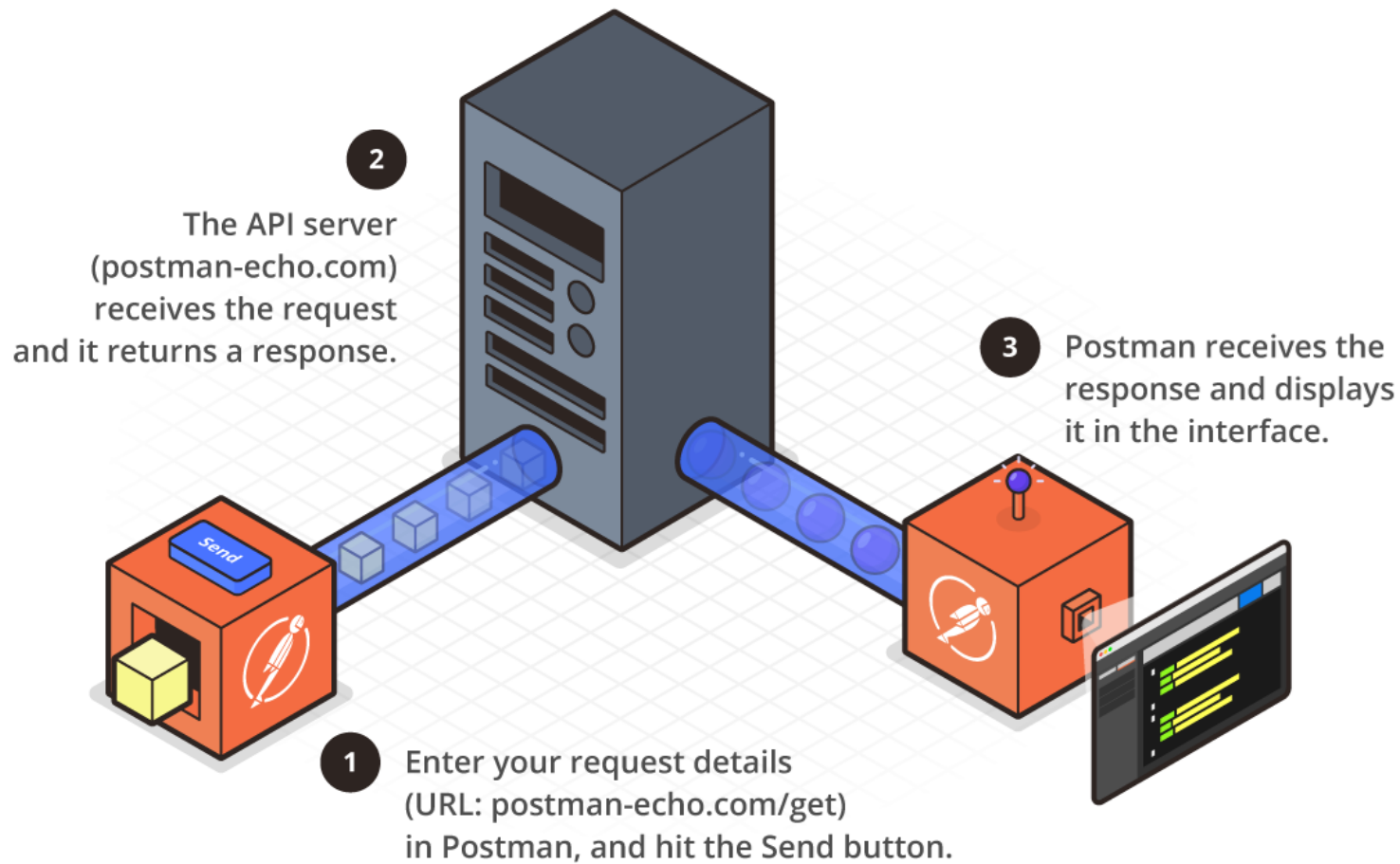
Query Params

Key	Value	Description	...	Bulk Edit
Key	Value	Description		

Body Cookies (2) Headers (6) Test Results Security 200 OK 124 ms 887 B Save as Example

Pretty Raw Preview Visualize JSON

```
1 {
2   "args": {},
3   "headers": {
4     "x-forwarded-proto": "https",
5     "x-forwarded-port": "443",
6     "host": "postman-echo.com",
7     "x-amzn-trace-id": "Root=1-645d2038-456598c75b724dbf2d1424e0",
8     "user-agent": "PostmanRuntime/7.32.1",
9     "accept": "*/*",
10    "cache-control": "no-cache",
11    "postman-token": "3eca8eeb-a246-4e63-bbe5-29cc6f14c9c1",
12    "accept-encoding": "gzip, deflate, br",
13    "cookie": "Cookie_2=value; sails.sid=s%3AA2wJ5pNb6IS6z9GBL7-0YJct0Wtnt7e6.N4WSeKB%2FMYaegxBz%2FhbxwtU1PJVahul0IoHEU248yFk"
14  },
15  "url": "https://postman-echo.com/get"
16 }
```



Recordando el ejemplo

```
from flask import Flask, request

app = Flask(__name__) #creamos una instancia de la clase Flask

@app.route("/signup/", methods=["GET"])
def metodo_que_procesa_peticion_get():
    return "No estas logeado. Esto es un get"

@app.route("/signup/", methods=[ "POST"])
def metodo_que_procesa_peticion_post():
    if request.is_json:
        data = request.get_json()
        name = data['name']
        email = data['email']
        password = data['password']
        return "Hola " + name + " " + email + " " + password
    return "No enviaste datos. Esto es un POST"

@app.route('/')
def index():
    return '<h1>Hola!</h1>'

if __name__ == '__main__':
    app.run(debug=True)
```




localhost:5000



Save

GET



localhost:5000

Send



Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

	Key	Value	Bulk Edit
	Key	Value	

Body

Cookies

Headers (5)

Test Results



200 OK

5 ms

187 B

Save Response



Pretty

Raw

Preview

Visualize

HTML



1 `<h1>Hola!</h1>`



localhost:5000



Save



Share

POST



localhost:5000

Send



Params Authorization Headers (8) Body Scripts Tests Settings

Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body Cookies Headers (6) Test Results



Status: 405 METHOD NOT ALLOWED

Time: 5 ms

Size: 370 B



Save as example



Pretty

Raw

Preview

Visualize

HTML



```
1 <!doctype html>
2 <html lang=en>
3 <title>405 Method Not Allowed</title>
4 <h1>Method Not Allowed</h1>
5 <p>The method is not allowed for the requested URL.</p>
```



localhost:5000/signup/



Save

GET



localhost:5000/signup/

Send



Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Cookies

Query Params

	Key	Value	Bulk Edit
	Key	Value	

Body

Cookies

Headers (5)

Test Results



200 OK

5 ms

205 B

Save Response



Pretty

Raw

Preview

Visualize

HTML



1 No estas logeado. Esto es un get



localhost:5000/signup

Save



Share

POST



localhost:5000/signup

Send



Params

Authorization

Headers (8)

Body

Scripts

Tests

Settings

Cookies

Query Params

	Key	Value	Description	...	Bulk Edit
	Key	Value	Description		

Body

Cookies

Headers (5)

Test Results



Status: 200 OK

Time: 5 ms

Size: 207 B



Save as example



Pretty

Raw

Preview

Visualize

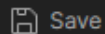
HTML



1 No enviaste datos. Esto es un POST



localhost:5000/signup



Save



Share

POST



localhost:5000/signup

Send



Params

Authorization

Headers (9)

Body

Scripts

Tests

Settings

Cookies

☐ none

☐ form-data

☐ x-www-form-urlencoded

☒ raw

☐ binary

☐ GraphQL

JSON



Beautify

```
1 {  
2   ... "name": "Damian",  
3     "email": "mail@mail.com",  
4   ... "password": "nolesvoyadecir"  
5 }
```

Body

Cookies

Headers (5)

Test Results



Status: 200 OK

Time: 7 ms

Size: 213 B



Save as example



Pretty

Raw

Preview

Visualize

HTML



```
1 Hola Damian mail@mail.com nolesvoyadecir
```

```
from flask import jsonify
# Base de datos simulada para propósitos de ejemplo
usuarios = {
    'usuario1@example.com': 'contrasena1',
    'usuario2@example.com': 'contrasena2'
}

@app.route('/login/', methods=['POST'])
def login():
    datos = request.get_json()
    email = datos['email']
    password = datos['password']

    if email in usuarios and usuarios[email] == password:
        respuesta = {'mensaje': 'Inicio de sesión exitoso'}
        codigo_respuesta = 200
    else:
        respuesta = {'mensaje': 'Credenciales incorrectas'}
        codigo_respuesta = 401

    return jsonify(respuesta), codigo_respuesta
```

localhost:5000/login/

Save

POST

localhost:5000/login/

Send

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

Cookies

none

form-data

x-www-form-urlencoded

raw

binary

JSON

Beautify

```
1
2  {
3    "email": "6536dfc62197bebffb91d7bb",
4    "password": 1
5  }
6
```

Body

Cookies

Headers (5)

Test Results



401 UNAUTHORIZED

15 ms

219 B

Save Response

Pretty

Raw

Preview

Visualize

JSON



```
1  {
2    "mensaje": "Credenciales incorrectas"
3  }
```



localhost:5000/login

Save

Share

POST

localhost:5000/login

Send

Params Authorization Headers (9) **Body** Scripts Tests Settings

Cookies

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

Beautify

```
1 {  
2   "email": "usuario1@example.com",  
3   "password": "contrasena1"  
4 }
```

Body Cookies Headers (5) Test Results

Status: 200 OK Time: 6 ms Size: 214 B Save as example

Pretty

Raw

Preview

Visualize

JSON



```
1 {  
2   "mensaje": "Inicio de sesión exitoso"  
3 }
```