

# Más de Backend

Cordero Hernández,  
Marco R.



# En sesiones pasadas

- Node.js
- npm
- Servicios Web / REST API
- Express
- Middlewares



# CONTENIDOS

**01**

Sistema de  
Archivos (fs)

**02**

Postman

**03**

Express Router

**04**

Páginas  
Estáticas

---

# 01

## SISTEMA DE ARCHIVOS (fs)



# Módulo de archivos *fs* y lectura

Para lectura y escritura de archivos (del lado del servidor) se cuenta *de manera nativa* el módulo **fs** (file system).

## Importación

- `const fs = require('fs');`

## Lectura

- Asíncrona: `readFile(archivo, callback(err, data))`
- Síncrona: `readFileSync(archivo)`

## Ejemplo

```
// Importar módulo
const fs = require('fs');

// Leer archivo (local)
let data = JSON.parse(fs.readFileSync('./fs1 - uso.json'));
```



# Escritura de archivos

- Asíncrona: `writeFile(archivo, datos, callback(err, data))`
- Síncrona: `writeFileSync(archivo)` // Si el archivo no existe, lo crea

## Ejemplo

```
// Importar módulo
const fs = require('fs');

// Datos a escribir
let newData = [
  { nombre: 'test1', carrera: 'ISC' },
  { nombre: 'test2', carrera: 'PSI' }
];
newData.push({ nombre: 'test3', carrera: 'ICD' });

// Escribir archivos (local)
fs.writeFileSync('datosAlumnos.txt', JSON.stringify(newData));
```

```
[marcordero@fedora Ejemplos]$ more datosAlumnos.txt
[{"nombre":"test1","carrera":"ISC"}, {"nombre":"test2","carrera":"PSI"}, {"nombre":"test3","carrera":"ICD"}]
```



# Lectura asíncrona

```
// Importar módulo
const fs = require('fs');

// Manejo de la lectura
let p = new Promise(resolve => {
  fs.readFile('datosAlumnos.txt', (err, data) => {
    if (err) {
      resolve('No fue posible abrir el archivo');
      return;
    }

    resolve(JSON.parse(data));
  });
});

// Muestra de los resultados
p.then(datos => console.log(datos));
```

```
[marcordero@fedora Ejemplos]$ node fs3\ -\ async.js
[
  { nombre: 'test1', carrera: 'ISC' },
  { nombre: 'test2', carrera: 'PSI' },
  { nombre: 'test3', carrera: 'ICD' }
]
```

---

# 02

## POSTMAN



# Introducción

**Postman** es una plataforma colaborativa para el desarrollo rápido y sencillo de **APIs**. Brinda soporte para diferentes clientes de **API** (pero solo se verá **REST**).

La plataforma inicialmente se desarrolló como extensión para los navegadores web, hoy en día ya cuenta con la opción de descarga como aplicación de escritorio.

Su uso es similar a la extensión *HTTP Rest Client* de VS Code, pero con muchas más funcionalidades.



# La polémica




Joel 🦀🐪 / based (same thing) ✓  
@ptr\_to\_joel

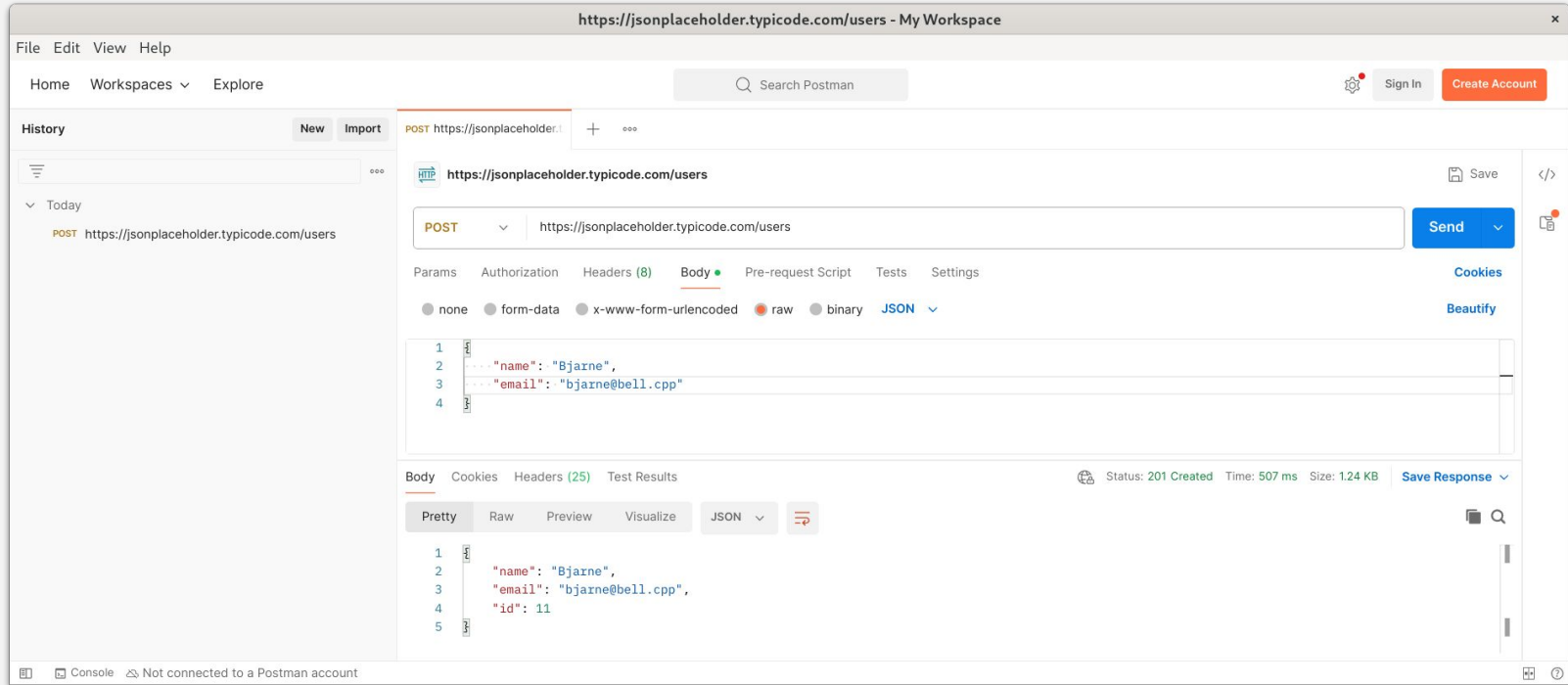
If a GUI wrapper around **curl** has the courage to host a conf, you can make your side projects public



**Postman** ✨ @getpostman · 27/02/24

We're offering 30% off tickets to early adopters until March 26 for POST/CON 24! 🚀 

# La polémica



# ***La polémica***

```
[marcordero@fedora ~]$ curl -X POST -d '{"name":"Bjarne","email":"bjarne@bell.cpp"}'  
-H "Content-Type: application/json" https://jsonplaceholder.typicode.com/users  
{  
  "name": "Bjarne",  
  "email": "bjarne@bell.cpp",  
  "id": 11  
}[marcordero@fedora ~]$
```

---

# 02.1

INSTALANDO  
POSTMAN





# Join 25 million developers who use Postman

Simplify your API design, development and testing workflows with Postman.

- ✦ Save and organize your work in collections and workspaces
- ✦ Share your work with clients and collaborate with teammates
- ✦ Integrate with tools like VS Code, GitHub and BitBucket



## Create a free Postman account

Sign up with email

Create Free Account

Already have an account? [Log In](#)

Or continue with the [lightweight API client](#).

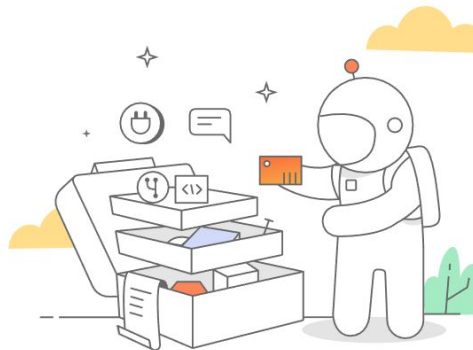


## Welcome to Postman! Tell us a bit about yourself.

Your name

What is your role?

Continue





Getting started - My Workspace

File Edit View Help

← → Home Workspaces API Network

Search Postman

Invite

Upgrade

Learn from experts and join the Postman community at POST/CON 24. Register by March 26 to save 30%.[Learn more](#)

My Workspace

New Import

Overview

Getting started

+

No environment

Collections

Environments

History

My first collection

First folder inside collection

GET

POST

GET

Second folder inside collection

GET

GET

Create a collection for your requests

A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.

Create Collection

Welcome to your personal workspace!

Here are some quick ways to get started with API development, design, testing, and more.

Send an API request

Ctrl+T

Import cURL, collections, and more

Ctrl+O

Recommended templates for getting started

[View all 70+ templates](#)

REST API basics

Get up to speed with testing REST APIs on Postman.

Integration testing basics

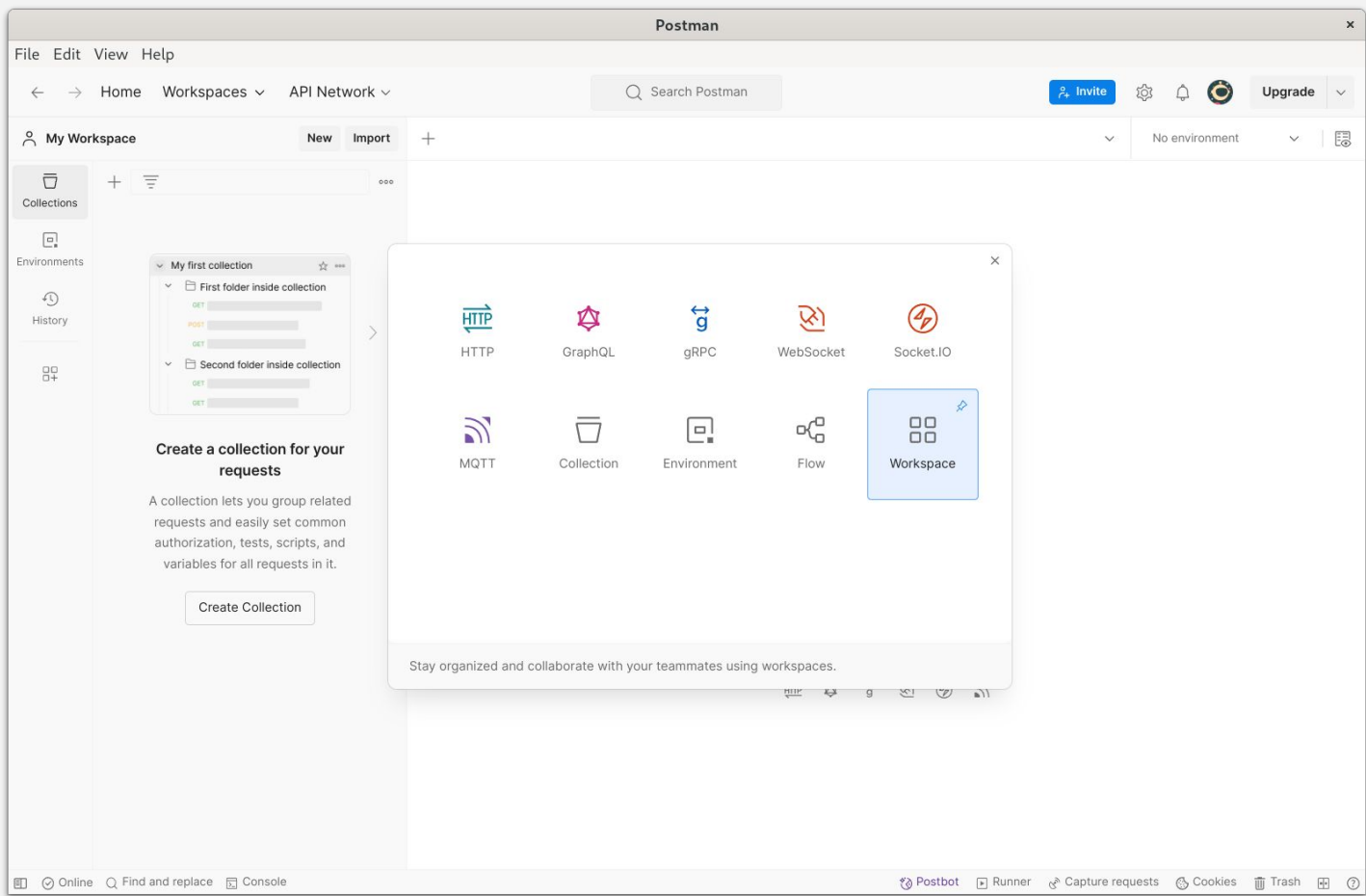
Verify if your APIs work as expected.

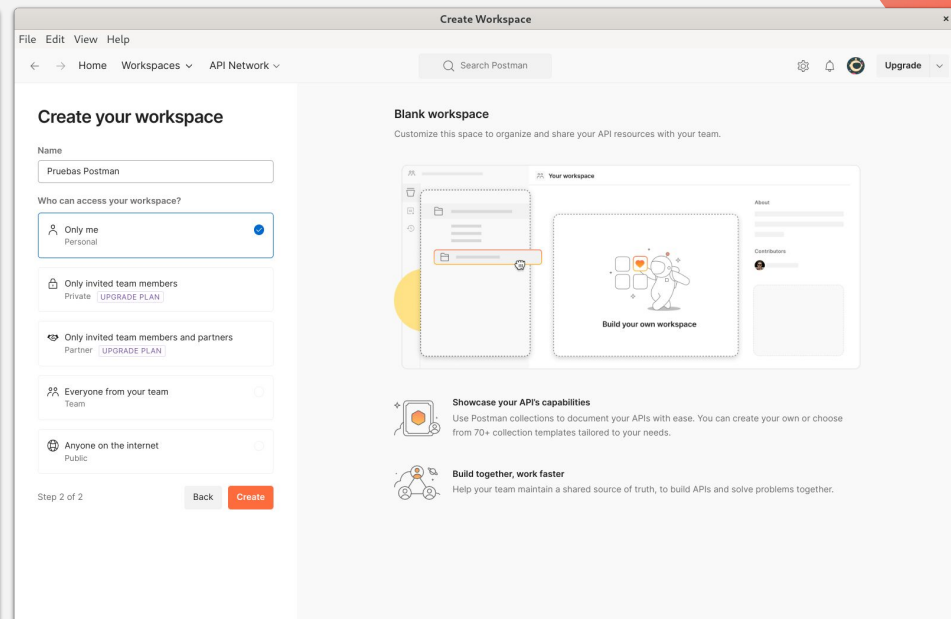
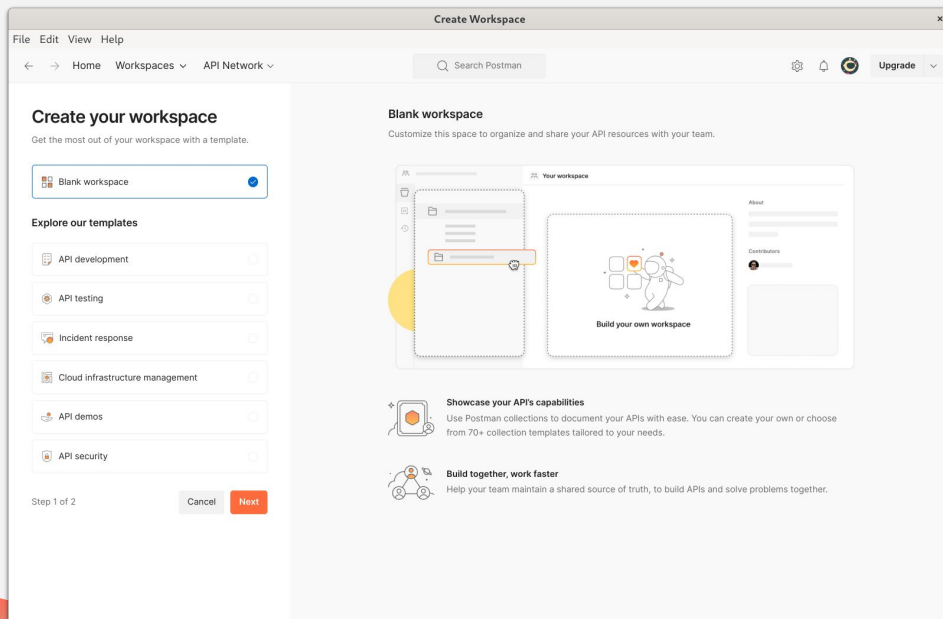
API documentation

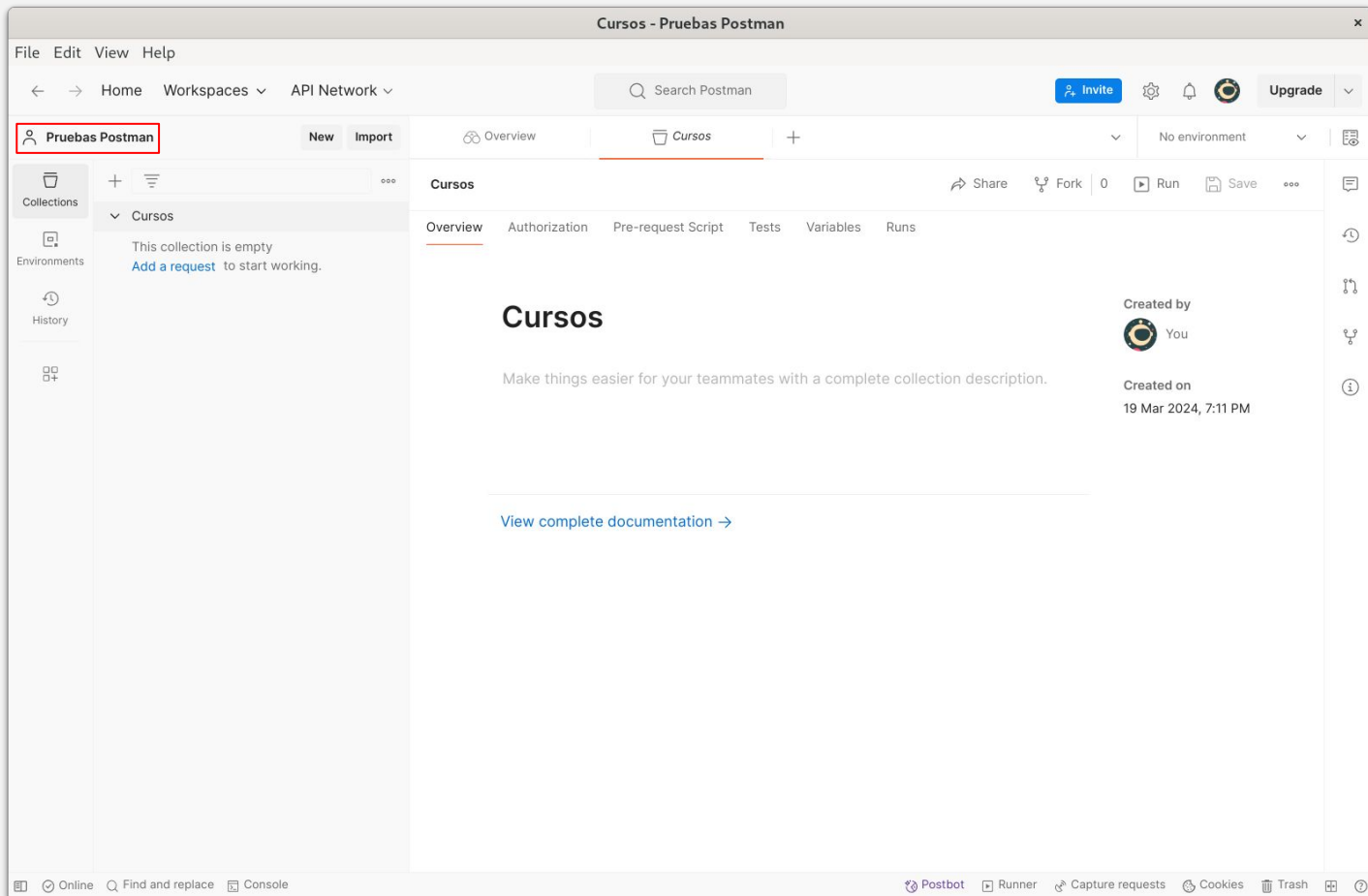
Create beautiful API documentation using Markdown.

Online Find and replace Console

Postbot Runner Capture requests Cookies Trash







FileEditViewHelp

←→HomeWorkspacesAPI Network

Search Postman

Invite

Upgrade

Pruebas Postman

NewImport

GET Petición GET

+

No environment

Cursos / Petición GET

Save

GET

https://jsonplaceholder.typicode.com/users

Send


ParamsAuthorizationHeaders (6)BodyPre-request ScriptTestsSettings

Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

Response



Click Send to get a response

OnlineFind and replaceConsole

PostbotRunnerCapture requestsCookiesTrash

Collections

Environments

History

Cursos

GET Pe

Share

Move

Run collection

Generate tests

BETA

Edit

Add request

Add folder

Monitor collection

Mock collection

Create a fork

Ctrl+Alt+F

Create pull request

Merge changes

Pull changes

View changelog

View documentation

Rename

Ctrl+E

Duplicate

Ctrl+D

Export



# Ejercicio

Desarrolla una **REST API** para el recurso de usuarios (users). Todos los endpoints estarán dentro de la ruta /api (ej. /api/users).

Para los métodos que reciban un cuerpo (body) de petición, usar el middleware **express.json()**.

Recurso	Endpoint	Métodos	Descripción
usuario	/api/users	GET, POST	Lista de usuarios, Nuevo usuario
	/api/users/{email}	GET, PUT, DELETE	Obtener usuario, Actualizar usuario, Borrar usuario

## Ejercicio – Express/REST API

1. Crea una ruta para **users** que muestre todos los usuarios (JSON completo)
2. Crea una ruta para **users/:email** (tal cual) que muestre el usuario con el email indicado (JSON completo también)
  - a. Apóyate de `req.params.email`
3. En el caso de **POST**, **PUT** y **DELETE** se debe mostrar un mensaje de texto con el nombre de usuario involucrado
  - a. Apóyate de `req.body`

## Ejercicio – FS/Postman

1. Completa el archivo **data\_handler.js** con la funcionalidad necesaria para leer y modificar correctamente los usuarios. La información se va a cargar y almacenar desde y hacia **users.json**
2. Especifica los **headers** correspondientes a cada ruta
3. Utiliza **Postman** para verificar que funciona la **REST API**



---

# 03

## EXPRESS ROUTER

## ***require / exports***

Para poder utilizar la funcionalidad de otros archivos de **JS**, se utilizará el mismo **require** usado para librerías de **npm**.

La instrucción **require** otorga el acceso al objeto **exports** del módulo que se está importando, por lo que en el archivo desde donde se desea exportar se *deben especificar qué cosas deben hacerlo*.

Es posible exportar todo el contenido del archivo, o solo unas partes.



# Uso de *require* / *exports*

```
function sumar(a,b) {  
    return a + b;  
}  
  
class AlgunaClase {  
    ...  
}  
  
// Ejemplo 1 - Todo el objeto  
module.exports = AlgunaClase;  
  
// Ejemplo 2 - Algunas propiedades  
exports.sumar = sumar;  
exports.clase = AlgunaClase;  
  
// Ejemplo 3 - Uso de objetos  
module.exports = {  
    sumFunc: sumar,  
    clase: AlgunaClase  
};
```

```
const importando = require('./modulo'); // No se necesita .js  
importando.sumFunc(1, 24); // 25  
  
// Importar solo unas funciones  
const { sumFunc } = require('./modulo');  
sumFunc(2, 5); // 7
```



# Router

Para dividir las rutas (**endpoints**) de alguna aplicación en varios archivos, *Express* cuenta con un objeto especial para *enrutar* recursos: **Router**. Su uso permite la fácil colaboración entre desarrolladores y evita archivo monolíticos.

```
// Rutas de usuario
const express = require('express');
const router = express.Router();

router.get('/', (req, res) => {
  res.send('Hola desde usuarios');
});

module.exports = router;
```

```
// Router global
const express = require('express');
const router = express.Router();
const users = require('./users');

router.get('/', (req, res) => {
  res.send('Ruta raíz');
});

// Uso del router
router.use('/users', users);

// Exportar router (usar desde app)
module.exports = router;
```

## Ejercicio

Modifica algún ejercicio realizado previamente para que las rutas de la aplicación estén definidas dentro de un **router.js**

Importa este nuevo módulo desde **server.js** o **index.js** con `app.use(router)` ;

---

# 04

PÁGINAS  
ESTÁTICAS



# Concepto y uso

Es posible utilizar el *middleware* **express.static()** que permita obtener archivos estáticos (html, css, javascript, imágenes, etc.) de un sitio web al acceder a una ruta determinada.

**ASÍ SE MANDA CONTENIDO DESDE UN SERVIDOR.**

```
// Archivos estáticos
app.use(express.static(`${__dirname}/public/home`));
app.use('/admin', express.static(`${__dirname}/public/admin`));

// Rutas
app.get('/generic', (req, res) => {
  let page = path.join(__dirname, 'public', 'generic.html');
  res.sendFile(page);
});
```