




# Node & NPM

 2025-01   30 min.

# Que es NodeJS?

Un *runtime environment* para JS.

# Que es un *Runtime Environment* ?

Coloquialmente, es donde corre un programa.

# Que es NPM?

Node Package Manager

# Que es un package?

Un “pequeño” software instalable/agregable a una aplicación de Node que agrega/modifica funcionalidad.

Todos los packages están listados en <https://www.npmjs.com/>

# Creando un proyecto de Node

## Normal

```
$ npm init
```

## Rapido

```
$ npm init --y
```

# Creando un proyecto de Node

```
1  {  
2    "name": "test",  
3    "version": "1.0.0",  
4    "description": "",  
5    "main": "index.js",  
6    "scripts": {  
7      "test": "node index"  
8    },  
9    "author": "",  
10   "license": "ISC",  
11  }
```

**Express**



# Un server sencillo

```
1  const http = require("http");
2
3  function requestListener(req, res) {
4    res.writeHead(200);
5    res.end("Hello, World!");
6  }
7
8  const server = http.createServer(requestListener);
9  server.listen(8080);
```

# Nodemon

Un package que nos permite hacer "hot reloading", es decir, que al realizar cambios el servidor automaticamente se reinicie.

# Nodemon?

Node ya implementó una manera de hacer esto sin instalar packages externos:

```
$ node --watch index.js
```

Sin embargo, Nodemon tiene otras funcionalidades.

# Instalar un package

```
$ npm i {nombre}
```

# Instalar nodemon

```
npm i nodemon
```

```
npm install nodemon
```

# Responder con JSON

```
1  const requestListener = function (req, res) {  
2    res.setHeader("Content-Type", "application/json");  
3    res.writeHead(200);  
4    res.end(`{"message": "This is a JSON response"}`);  
5  };
```

# JSON

Java Script Object Notation

O, Notación de Objeto de JavaScript.

Es una manera de representar (en una string) un objeto de JavaScript. Típicamente se espera enviar y recibir datos usando JSON.

# Express

```
$ npm i express
```



# Inicialización de un servidor de Express

```
1  const express = require("express");  
2  const app = express();
```

# Un Endpoint en Express

```
1 app.get("/", async function (req, res) {  
2   res.status(200).json({ message: "Success" });  
3 });
```

# Escuchar por peticiones en Express

```
1  const port = 3000;
2
3  app.listen(port, () => {
4    console.log(`Example app listening on port ${port}`);
5  });
```

# Operaciones basicas de almacenamiento

CRUD

# Operaciones basicas de almacenamiento

1. C

2. R

3. U

4. D

# Operaciones basicas de almacenamiento

1. C

2. R

3. U

4. creaDo

# Operaciones basicas de almacenamiento

1. C
2. R
3. hUsmear
4. creaDo

# Operaciones basicas de almacenamiento

1. aCtualizar
2. R
3. hUsmear
4. creaDo



# Operaciones basicas de almacenamiento

1. aCtualizar
2. borRar
3. hUsmear
4. creaDo

# Operaciones basicas de almacenamiento

1. **Create**
2. **Read**
3. **Update**
4. **Delete**

# Metodos HTTP

GET	HEAD	POST
PUT	DELETE	CONNECT
OPTIONS	TRACE	PATCH

# Metodos HTTP

Tipicamente usados en REST:

GET

PUT

POST

PATCH

DELETE

---

No tan tipicos:

OPTIONS

TRACE

HEAD

CONNECT

---

# Metodos HTTP => CRUD

GET	PUT	POST	PATCH	DELETE
READ	UPDATE/CREATE	CREATE	UPDATE	DELETE

# Put vs Patch

Put sobre-escribe (o crea) un recurso.

Patch modifica un recurso ya existente.

# Metodos HTTP => CRUD

GET	PUT	POST	PATCH	DELETE
READ	UPDATE/CREATE	CREATE	UPDATE	DELETE

# CORS

## Cross-Origin Resource Sharing

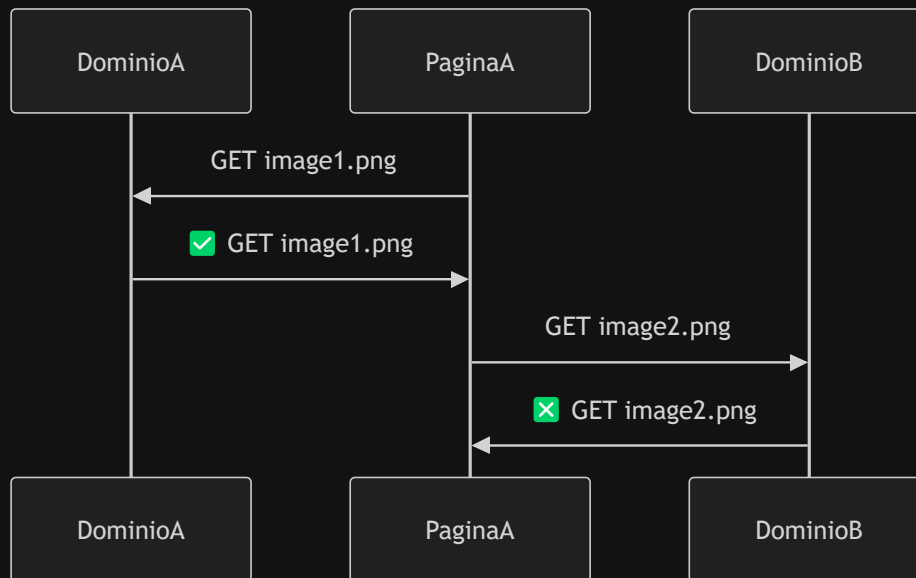
1. Medida de seguridad para prevenir abuso.
2. Previene utilizar recursos de otro origen a menos que este lo permita.



# CORS

✖ Access to XMLHttpRequest at '[http://localhost:5000/global\\_config](http://localhost:5000/global_config)' step1:1 from origin '<http://localhost:8080>' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource.

# CORS



# Metodo `OPTIONS` + CORS

El metodo `OPTIONS` tiene como proposito preguntar los permisos de CORS al origen del recurso previo a solicitar el recurso.

Un problema de CORS 99% de las veces es un problema de backend.

# CORS

Un navegador hace un metodo `OPTIONS` para saber los permisos.

Un cliente REST NO.

Clientes REST como Postman, ThunderClient, Insomnia, etc.

# CORS

Front end dev: "Este endpoint me tira error."

Back end dev: "Pero lo pruebo en Postman y me funciona sin problema."

Conclusión: Revisa CORS.

# CORS - Solución

```
1  app.use(function (req, res, next) {
2    res.header("Access-Control-Allow-Origin", "*");
3    res.header("Access-Control-Allow-Headers", "*");
4
5    if (req.method === "OPTIONS") {
6      res.header("Access-Control-Allow-Methods", "PUT, POST, PATCH, DELETE, GET");
7      return res.status(200).json({});
8    }
9
10   next();
11 });
```

Agregar este endpoint al inicio del server responde toda petición de `OPTIONS` con acceso permitido para todo.

# CORS - Solución

```
$ npm i cors
```

Este package hace todo lo que hace el endpoint anterior.

# CORS - Solución 2

```
1  let express = require("express");
2  let cors = require("cors");
3  let app = express();
4
5  app.use(cors());
6
7  app.get("/products/:id", function (req, res, next) {
8    res.json({ msg: "CORS resuelto!" });
9  });
```



# "Fallbacks"

```
1 app.use(async function (req, res) {  
2   res.status(404).json({ message: "Not found." });  
3 });
```

Este endpoint va al final, para recibir toda petición que no fue servida por otros endpoints.

```
1  // imports
2  const express = require("express");
3  const cors = require("cors");
4  const app = express();
5  const port = 3000;
6
7  // resuelve CORS
8  app.use(cors());
9
10 // un endpoint GET
11 app.get("/products/:id", function (req, res, next) {
12   res.json({ msg: "CORS resuelto!" });
13 });
14
15 // endpoint de 404
16 app.use(async function (req, res) {
17   res.status(404).json({ message: "Not found." });
18 });
19
20 // 'iniciar' servidor
21 app.listen(port, () => {
22   console.log(`Example app listening on port ${port}`);
23 });
```



**Saben Node y Express!**