



## Clase 6. Programación Backend

# ***Servidores Web***



## ***OBJETIVOS DE LA CLASE***

- Creación de un servidor web usando el módulo HTTP.
- Creación de un servidor web usando el módulo Express.
- Realizar el despliegue de nuestra aplicación backend en la nube.

# ***CRONOGRAMA DEL CURSO***

Clase 5



**Administradores de  
Paquetes - NPM**

---

---

---

---

Clase 6



**Servidores Web**

Clase 7



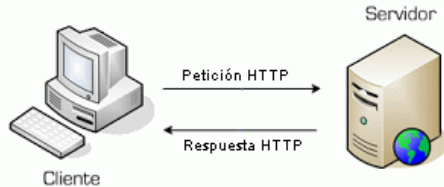
**Express Avanzado**

---

---

---

---



# Nuestro primer servidor HTTP



## NodeJs en Servidor

### Ejemplo "Hola Mundo" en la Web

```
const http = require('http');
const server = http.createServer();
server.on('request', procesa);
server.listen(3000);
console.log('Servidor arrancado');

function procesa(request, response) {
  let url = request.url;
  console.log(`URL solicitada: ${url}`);
  response.end("Hola");
}

$ node hola.js
Servidor arrancado
```

```
const server = http.createServer(function (req, res) {
  res.writeHead(200, {'content-type': 'text/plain'});
  res.end('Hola Mundo');
});
```

# ***Servicios Web***

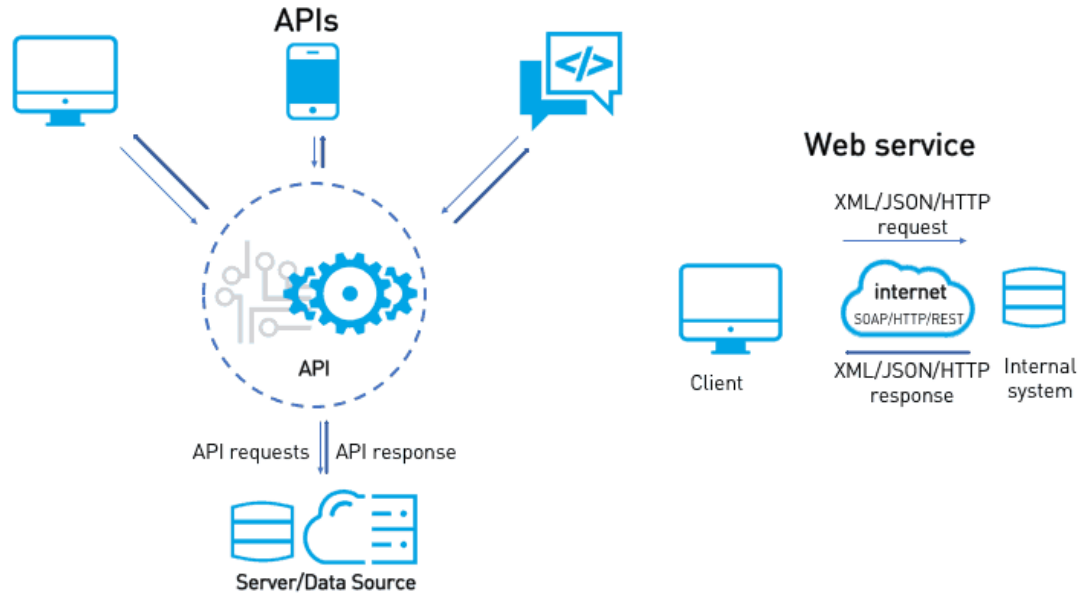
# ***Servicios Web***



- Un web service o servicio web es un tipo de tecnología que, a través de ciertos protocolos y estándares, habilita la comunicación entre distintas computadoras y permite intercambiar datos entre ellas.

\*Los más conocidos REST, SOAP

# ***Servicios Web***





# ***Instalar Nodemon***

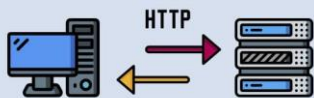


- Vamos a instalar la dependencia **nodemon** de forma global usando **npm**.
- **Nodemon** nos ayuda en el desarrollo relanzando la ejecución de Node.js en el caso de que algún archivo de nuestro proyecto cambie.
- Instalamos la librería desde una terminal ejecutando: `npm i -g nodemon`



# ***Módulo HTTP***

- HTTP es un **módulo nativo** de Node.js
- Trabaja con el **protocolo HTTP**, que es el que se utiliza en Internet para transferir datos en la Web.
- Nos va a servir para **crear un servidor HTTP** que acepte solicitudes desde un cliente web.
- Para poder utilizarlo en nuestro código, tenemos que requerirlo mediante la instrucción `require( 'http' )` y guardarlo en una variable para su posterior uso.

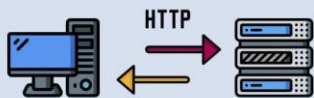


# ***Servidor HTTP paso a paso***



```
const http = require('http')
```

- A partir de este momento tenemos una **variable http** (que en realidad es un objeto) sobre la que **podemos invocar métodos** que estaban en el módulo requerido.
- Por ejemplo, una de las tareas implementadas en el módulo HTTP es la de **crear un servidor**, que se hace con el módulo "createServer()".
- Este método recibirá un callback que se ejecutará cada vez que el servidor reciba una petición.

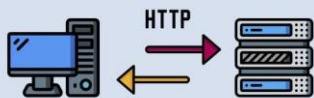


# ***Servidor HTTP paso a paso***



```
const server = http.createServer((peticion, respuesta) => {  
  respuesta.end('Hola mundo')  
})
```

- La función **callback** que enviamos a `createServer()` **recibe** dos parámetros que son la **petición** y la **respuesta**.
- La petición por ahora no la usamos, pero contiene datos de la petición realizada.
- La respuesta la usaremos para enviarle datos al cliente que hizo la petición.
- De modo que "**respuesta.end()**" sirve para **terminar** la **petición** y **enviarle datos al cliente**.

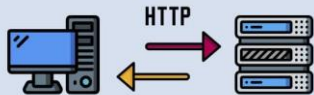


# ***Servidor HTTP paso a paso***



```
const connectedServer = server.listen(8080, () => {  
  console.log(`Servidor Http escuchando en el puerto ${connectedServer.address().port}`)  
})
```

- Con esto le decimos al **servidor** que **escuche** en el **puerto 8080**, aunque podríamos haber puesto cualquier otro puerto que nos hubiera gustado.
- "**listen()**" **recibe** también una **función callback** que realmente no sería necesaria, pero que nos sirve para hacer cosas cuando el servidor se haya iniciado y esté listo. Simplemente, en esa función callback **indico** que estoy **listo** y **escuchando** en el **puerto configurado**.
- Listen, además, devuelve un objeto que contiene los datos del servidor conectado.



# ***Servidor HTTP paso a paso***

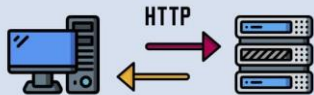


```
const http = require('http')

const server = http.createServer((peticion, respuesta) => {
  respuesta.end('Hola mundo')
})

const connectedServer = server.listen(8080, () => {
  console.log(`Servidor Http escuchando en el puerto ${connectedServer.address().port}`)
})
```

- Este es el código completo. En muy pocas líneas de código **generamos un servidor web que está escuchando en un puerto dado**. Ahora podemos guardar ese archivo con extensión .js, por ejemplo “servidor.js”.



# *Configurando EndPoints*



```
const server = http.createServer((peticion, respuesta) => {  
  if (peticion.url == '/') { //Evalua la URL solicitada  
    respuesta.writeHead(200, { 'Content-Type': 'text/html' });  
    respuesta.write('<html><body><p>This is home Page. yeah!</p></body></html>');  
    respuesta.end();  
  }  
  else if (peticion.url == "/productos") {  
    . . .  
  }  
})
```

- El objeto **peticion** posee la propiedad **url** que nos brinda la posibilidad de configurar los endpoints de la API.

# ***Ahora: ¡Poner en ejecución el archivo con Node.JS para iniciar el servidor!***

1. Vamos desde la línea de comandos a la carpeta donde guardamos el archivo *servidor.js* y ejecutamos el comando "node" seguido del nombre del archivo que pretendemos ejecutar: ***node servidor.js***
2. En la consola de comandos aparecerá el mensaje que informa que nuestro servidor está escuchando en el puerto 8080.
3. El modo de comprobar si realmente el servidor está escuchando a solicitudes de clientes en dicho puerto es acceder con un navegador a la dirección:  
<http://localhost:8080>
4. En la vista del navegador se mostrará el mensaje "*Hola mundo!*" devuelto por el servidor.



# ***Servidor en Node***



# MENSAJE SEGÚN LA HORA

Desafío  
generico



Desarrollar un servidor en node.js que escuche peticiones en el puerto 8080 y responda un mensaje de acuerdo a la hora actual:

- Si la hora actual se encuentra entre las 6 y las 12 hs será 'Buenos días!'.
- Entre las 13 y las 19 hs será 'Buenas tardes!'.
- De 20 a 5 hs será 'Buenas noches!'.

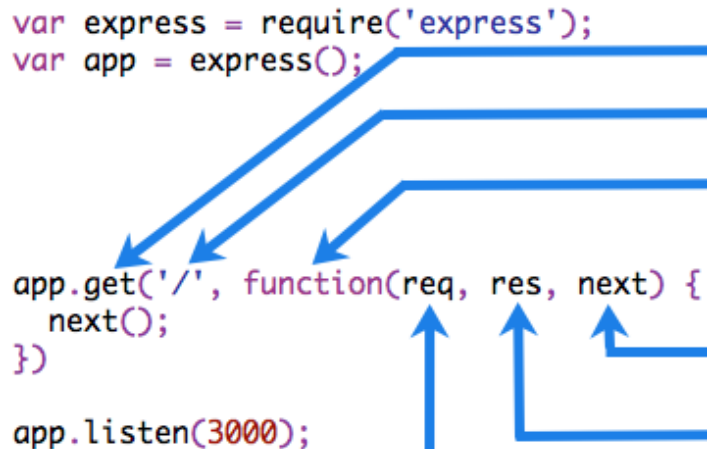
Se mostrará por consola cuando el servidor esté listo para operar y en qué puerto lo está haciendo.

*Tiempo: 5/10 minutos*

***CODER HOUSE***

# *Implementación de un servidor http en Express*

```
var express = require('express');  
var app = express();  
  
app.get('/', function(req, res, next) {  
  next();  
})  
  
app.listen(3000);
```

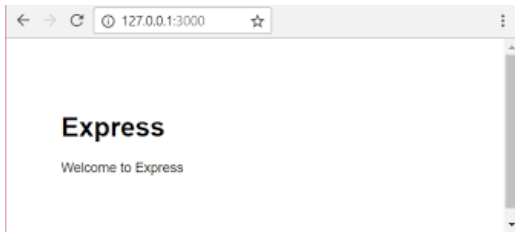


# ***Introducción***

NodeJS cuenta con módulos nativos para manejar el envío y recepción de peticiones de tipo http/s, sin embargo, usaremos para nuestra aplicación un módulo externo llamado **express**

*Algunas de sus principales características son:*

- ★ Es muy popular y fácil de usar.
- ★ Nos facilitará la tarea de **crear** los distintos **puntos de entrada** de nuestro **servidor**.
- ★ También permite personalizar la manera en que se maneja cada petición en forma más simple y rápida.



# ***Express.js***



Express es un **framework web** minimalista, con posibilidad de ser utilizado tanto para aplicaciones/páginas web como para aplicaciones de servicios. Como todo módulo, lo primero que debemos realizar es agregarlo como dependencia en nuestro proyecto

Instalación desde la consola:  
***npm install express***



Express

**JS**

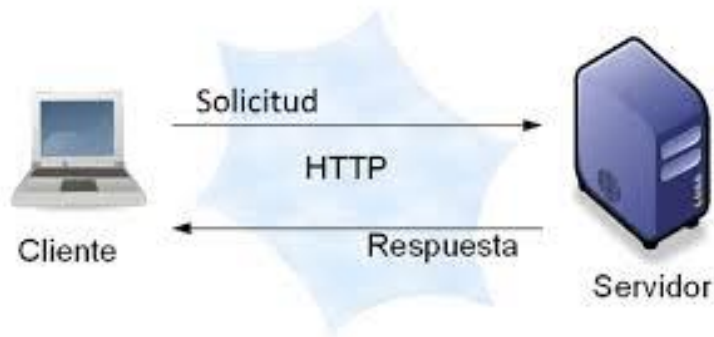
# ***Express como framework soporte para servidores REST***



# ***Introducción***



**Express** nos permite definir, para cada tipo de petición HTTP que llegue a una determinada URL, qué acciones debe tomar, mediante la definición de un callback para cada caso que consideremos necesario incluir en nuestra API.



## *Uso del módulo*

Para poder usar el módulo, lo primero que debemos hacer es **importarlo** al comienzo de nuestro archivo. El objeto obtenido luego del import es una **función**. Al ejecutarla, nos devolverá la **aplicación** servidor que configuraremos posteriormente con los detalles de nuestra aplicación.

### *Ejemplo de inicialización*

```
const express = require('express')  
  
const app = express()
```

# ***Conexión del servidor***

Debemos indicar en qué puerto de nuestra computadora queremos que nuestra aplicación comience a escuchar peticiones. Este puerto será de uso exclusivo de nuestro servidor, y no podrá ser compartido con otras aplicaciones.

## ***Ejemplo de conexión***

```
const PORT = 8080

const server = app.listen(PORT, () => {
  console.log(`Servidor http escuchando en el puerto ${server.address().port}`)
})
```

*Si el puerto elegido es el cero (0), express elegirá un puerto al azar entre los disponibles del sistema operativo en ese momento.*



# ***Manejo de errores de conexión***

Para indicar una situación de error en la puesta en marcha del servidor, podemos configurar el evento 'error' a través del método 'on' sobre la salida de 'listen'

## ***Ejemplo de conexión (con evento de error)***

```
const PORT = 8080

const server = app.listen(PORT, () => {
  console.log(`Servidor http escuchando en el puerto ${server.address().port}`)
})

server.on("error", error => console.log(`Error en servidor ${error}`))
```

*El **argumento error** del callback configurado para el **evento error**, nos da la descripción del error ocurrido.*

## ***Configuración petición Get***

Cuando queremos obtener algún tipo de información del servidor utilizamos peticiones de tipo **GET**. Este tipo de peticiones son las más comunes. Entonces, configuraremos en nuestro servidor un manejador para estas peticiones. Como respuesta, devolveremos el **resultado** deseado en **forma de objeto**.

### ***Ejemplo de manejador de peticiones GET a la ruta raiz del servidor***

```
app.get('/', (req, res) => {  
  res.send({ mensaje: 'hola mundo' })  
})
```

*req = request (petición) / res = response (respuesta)*



# ***Servidor con express***

Crear un proyecto de servidor http en node.js que utilice la dependencia express

*Tiempo: 10/15 minutos*

***CODER HOUSE***



Crear un proyecto de servidor http en node.js que utilice la dependencia express, escuche en el puerto 8080 y tenga tres rutas get configuradas:

**A)** '/' en esta ruta raíz, el servidor enviará string con un elemento de título nivel 1 (un h1 en formato HTML) que contenga el mensaje: 'Bienvenidos al servidor express' en color azul.

**B)** '/visitas' donde con cada request, el servidor devolverá un mensaje con la cantidad de visitas que se hayan realizado a este endpoint. Por ej. 'La cantidad de visitas es 10'

**C)** '/fyh' donde se devolverá la fecha y hora actual en formato objeto:  
{ fyh: '11/1/2021 11:36:04' }

Mostrar por consola el puerto de escucha del servidor al momento de realizar el listen.

En caso de error, representar el detalle.



***BREAK***

**¡5/10 MINUTOS Y VOLVEMOS!**

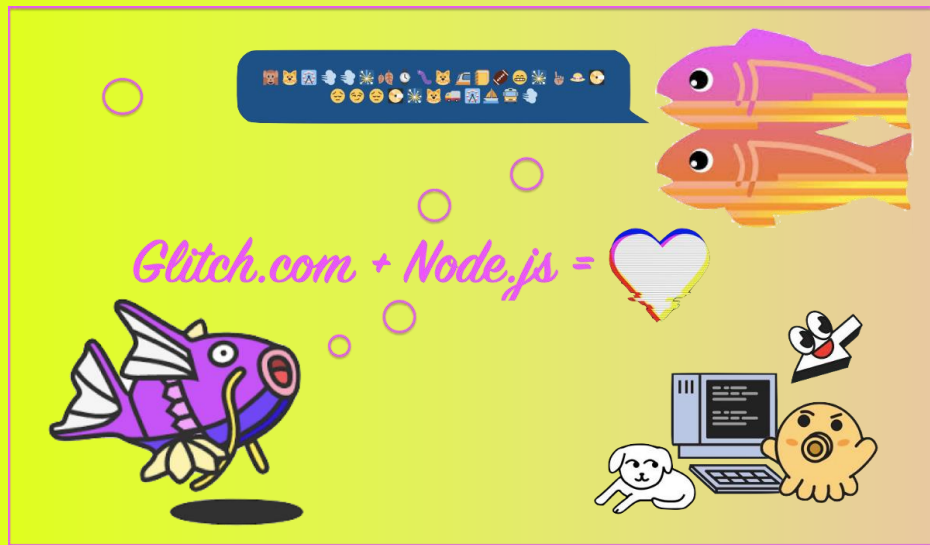
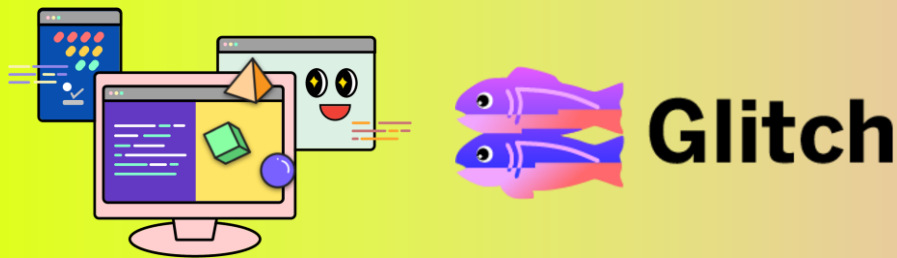


# ***Nuestro primer Despliegue en un servidor en la nube***

# ***Despliegue en la nube***

- ❑ Tomaremos el servidor del desafío anterior y lo desplegaremos en la nube.
- ❑ Utilizaremos el servicio gratuito provisto por [Glitch.com](https://glitch.com).
- ❑ Nos basaremos en un proyecto base provisto por el sitio, que contiene algunas carpetas y archivos para arrancar un proyecto ExpressJS desde cero. Podemos acceder a este proyecto en la siguiente URL:  
<https://glitch.com/edit/#!/hello-express>
- ❑ Es necesario crear una cuenta (gratuita) para que nuestros proyectos persistan por más de 5 días desplegados. Utilizaremos esta funcionalidad en próximos desafíos durante el curso.

# ***Despliegue del servidor en glitch.com***



**CODER HOUSE**



# Code, collaborate, & ship in seconds

Create your next web project in your browser with no setup and we'll instantly deploy it.

[Join for free](#)

## Less researching, more coding

We picked the best tools, libraries and frameworks so you don't have to. Get started now with one of our project templates.



# There's always something new to make on Glitch

Our Hello apps are built with best practices in mind and sample code you can remix to make your own. Or grab a minimal template and build from a blank slate.



## Basic Website

A simple website starter

[Hello Website](#)

[blank version →](#)



## Glitch in Bio

Your own free links page

[Remix your own](#)

[learn more →](#)



## Hello Node!

Full stack, ready to go

[Hello Node](#)

[blank version →](#)



## Blog with Eleventy

As easy as blogging gets

[Hello Eleventy](#)

[minimal 11ty \(by 11ty\) →](#)




Loading Project



*Glitch Tip!*

Drag in images and files to add them to your project

This site is protected by reCAPTCHA and the Google [Privacy Policy](#) and [Terms of Service](#) apply.

Glitch

README.md

Remix this projectLog inSign up

nonstop-tangy-observation<<README.mdEDIT MARKDOWN

SettingsAssetsFiles+> public/> src/.env.gitignoreLICENSEREADME.mdpackage.jsonserver.js

## Hello Node (blank)

[Node.js](#) is a popular runtime that lets you run JavaScript on the server. This project uses the [Fastify](#) framework and basic templating with [Handlebars](#).

### What's in this project?

← `README.md`: That's this file, where you can tell people what your cool website does and how you built it.


← `public/style.css`: The styling rules for your pages and posts.

← `server.js`: The main server script for your new site.

← `src/`: This folder holds page templates, additional scripts.

### Working in the `src/` folder

← `src/pages/index.hbs`: This is the main page template for your site.



### You built this with Glitch!

[Glitch](#) is a friendly community where millions of people come together to build web apps and websites.

- Need more help? [Check out our Help Center](#) for answers to any common questions.
- Ready to make it official? [Become a paid Glitch member](#) to boost your app with private sharing, more storage and memory, domains and more.

/

## Hello Node!

This starter gives you everything you need to start working on a new project using Node. If you're looking for something a little more opinionated, try remixing [~glitch-hello-node](#).

STATUSLOGSTERMINALTOOLSPREVIEW

nonstop-tangy-observation &lt;&lt;

server.js

PRETTIER

Settings

Assets

Files

&gt; public/

&gt; src/

.env

.gitignore

LICENSE

README.md

package.json

server.js

```
1 const express = require('express')
2 const PORT = 8080;
3
4 const app = express();
5
6 app.get('/', (req,res) => {
7   res.send('<h1 style="color:blue;">Bienvenidos al servidor express</h1>')
8 })
9
10 app.get('/student', (req,res) => {
11   res.send('<html><body><p>This is student Page.</p></body></html>');
12 })
13
14 app.get('/admin', (req,res) => {
15   res.send('<html><body><p>This is admin Page.</p></body></html>');
16 })
17
18 const server = app.listen(PORT, () => {
19   console.log(`Servidor http escuchando en el puerto ${server.address().port}`)
20 })
21 server.on("error", error => console.log(`Error en servidor ${error}`));
22
```

## Hello Node!

This starter gives you everything you need to start working on a new project using Node. If you're looking for something a little more opinionated, try remixing [-glitch-hello-node](#).



STATUS



LOGS



TERMINAL



TOOLS



PREVIEW



Glitch

server.js

Remix this project

It looks like you just made this project a static site. Did you know that static sites stay awake and are always on for free? [Learn More](#) [Close](#)

nonstop-tangy-observation

server.js

PRETTIER

Settings

Assets

Files

public/

src/

.env

.gitignore

LICENSE

README.md

server.js

```
1 const express = require('express')
2 const PORT = 8080;
3
4 const app = express();
5
6 app.get('/', (req,res) => {
7   res.send('<h1 style="color:blue;">Bienvenidos al servidor express</h1>')
8 })
9
10 app.get('/student', (req,res) => {
11   res.send('<html><body><p>This is student Page.</p></body></html>');
12 })
13
14 app.get('/admin', (req,res) => {
15   res.send('<html><body><p>This is admin Page.</p></body></html>');
16 })
17
18 const server = app.listen(PORT, () => {
19   console.log('Servidor http escuchando en el puerto ${server.address().port}')
20 })
21 server.on("error", error => console.log('Error en servidor ${error}'));
22
```

Search

node\_modules

public

src

LICENSE

Terminal

Full Page Terminal

```
"test": "echo \"Error: no test specified\" && exit 1",
"start": "node server.js"
},
"author": "",
"license": "ISC"
}

Is this OK? (yes)

app@nonstop-tangy-observation:~ 00:10
$ npm install express
```

STATUS

LOGS

TERMINAL

TOOLS

PREVIEW

<https://glitch.com/>

**CODER HOUSE**

Glitch

server.js

Remix this projectLog inSign up

nonstop-tangy-observation<<server.jsPRETTIER

SettingsAssetsFiles+> public/> src/.env.gitignoreLICENSEREADME.mdserver.js

```
1 const express = require('express')
2 const PORT = 8080;
3
4 const app = express();
5
6 app.get('/', (req,res) => {
7   res.send('<h1 style="color:blue;">Bienvenidos al servidor express</h1>')
8 })
9
10 app.get('/student', (req,res) => {
11   res.send('<html><body><p>This is student Page.</p></body></html>');
12 })
13
14 app.get('/admin', (req,res) => {
15   res.send('<html><body><p>This is admin Page.</p></body></html>');
16 })
17
18 const server = app.listen(PORT, () => {
19   console.log('Servidor http escuchando en el puerto ${server.address().port}')
20 })
21 server.on("error", error => console.log('Error en servidor ${error}'));
22
```

Search

node\_modulespublicsrcLICENSE

TerminalFull Page Terminal →

```
app@nonstop-tangy-observation:~ 00:11
$ npm install
npm WARN app@1.0.0 No repository field.

added 93 packages from 100 contributors, removed 17 packages, moved 24 packages and audited 240 packages in 22.801s
found 0 vulnerabilities

app@nonstop-tangy-observation:~ 00:12
$ node server.js
Servidor http escuchando en el puerto 8080

```

STATUSLOGSTERMINALTOOLSPREVIEW



# ***SERVIDOR CON EXPRESS***

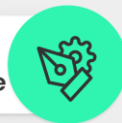


# ***Servidor con express***

**Formato:** link a un repositorio en Github y url de proyecto subido a glitch

**Observación:** no incluir la carpeta *node\_modules*

Desafío  
entregable



## **>> Consigna:**

- 1) Realizar un proyecto de servidor basado en node.js que utilice el módulo express e implemente los siguientes endpoints en el puerto 8080:
  - a) Ruta get '/productos' que devuelva un array con todos los productos disponibles en el servidor
  - b) Ruta get '/productoRandom' que devuelva un producto elegido al azar entre todos los productos disponibles
- 2) Incluir un archivo de texto 'productos.txt' y utilizar la clase Contenedor del desafío anterior para acceder a los datos persistidos del servidor.

Antes de iniciar el servidor, colocar en el archivo 'productos.txt' tres productos como en el ejemplo del desafío anterior.

# ***Servidor con express***

**Formato:** link a un repositorio en Github y url de proyecto subido a glitch

**Observación:** no incluir la carpeta *node modules*

Desafío  
entregable



```
[
  {
    "title": "Escuadra",
    "price": 123.45,
    "thumbnail": "https://cdn3.iconfinder.com/data/icons/education-209/64/ruler-triangle-stationary-school-256.png",
    "id": 1
  },
  {
    "title": "Calculadora",
    "price": 234.56,
    "thumbnail": "https://cdn3.iconfinder.com/data/icons/education-209/64/calculator-math-tool-school-256.png",
    "id": 2
  },
  {
    "title": "Globo Terráqueo",
    "price": 345.67,
    "thumbnail": "https://cdn3.iconfinder.com/data/icons/education-209/64/globe-earth-geograhpy-planet-school-256.png",
    "id": 3
  }
]
```

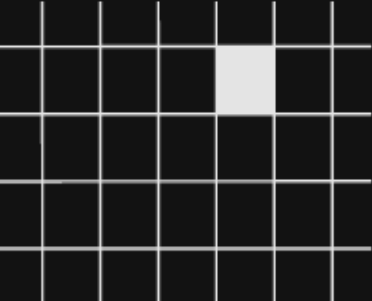
***¿PREGUNTAS?***





# ***¡MUCHAS GRACIAS!***

Resumen de lo visto en clase hoy:

- Servidores con módulo HTTP y Express
  - Despliegue en Glitch
- 



***OPINA Y VALORA ESTA CLASE***

***#DEMOCRATIZANDOLAEDUCACIÓN***