



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

Repasando...

- Recordemos que **NPM** es el **Package Manager** (Administradores de paquetes) de **Node Js**.
- Es una **aplicación** que facilita la **descarga** e **instalación** de **módulos** que necesita tu proyecto.
- NPM nos permite **instalar** las **dependencias** de nuestro proyecto desde el archivo **package.json**. Esta instalación es local.
- También nos permite **instalar** **dependencias globales** que podemos usar de forma **transversal** entre nuestros proyectos, normalmente **durante el desarrollo**.

NPM

- Las dependencias son **módulos de terceros** que necesita nuestro proyecto para poder **ejecutarse** o **construirse**.
- Esto nos indica que tenemos **dos tipos de dependencias**, las que usa nuestro proyecto cuando se encuentra en un **ambiente de ejecución**, y las de desarrollo, que se utilizan **durante el desarrollo**.
- Debemos tener presente las **versiones** de nuestras dependencias, ya que estas pueden sufrir **cambios** por **mejoras** o **corrección** de **vulnerabilidades**, las cuales pueden **afectar el correcto funcionamiento** de nuestro proyecto.

Dependencias

- Nos permite determinar el tipo de actualización que recibe una dependencia a través de un grupo de tres cifras.
- Las actualizaciones de menor impacto, normalmente corrección de errores compatibles con versiones anteriores.
- Las actualizaciones de medio impacto, donde los cambios son nuevas funciones o mejoras compatibles con versiones anteriores.
- Finalmente las de mayor impacto, que se caracteriza por no ser compatible al 100% con versiones anteriores.

Versionado Semántico

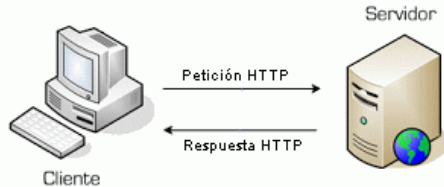


- Ya sabemos que las **dependencias** reciben **actualizaciones** y debemos estar atento a estas y a la **forma** en que nuestro proyecto las **maneja**. Esto nos lo permiten los siguientes **símbolos** junto a su versión.
- **~0.13.0** actualizaciones de corrección de errores compatibles con versiones anteriores.
- **^0.13.0** actualizaciones de nuevas funciones o mejoras compatibles con versiones anteriores.
- ***0.13.0** actualizaciones que no se garantiza la compatibilidad con versiones anteriores.
- **0.13.0** finalmente si no especifica el símbolo, la versión será fija.

Actualizaciones de dependencias en Node Js



***¿Alguna pregunta hasta
ahora?***



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');
});
```



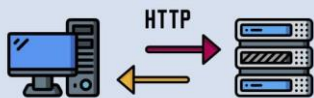
Instalar Nodemon



- 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, lo debemos requerir mediante la instrucción `require('http')` y guardarlo en una variable para su posterior uso.

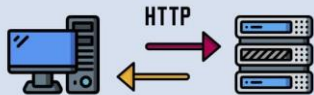


Servidor HTTP paso a paso



```
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.

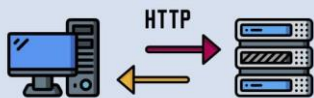


Servidor HTTP paso a paso



```
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**.

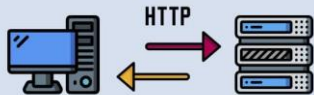


Servidor HTTP paso a paso



```
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”.

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.



***¿Alguna pregunta hasta
ahora?***



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!'.

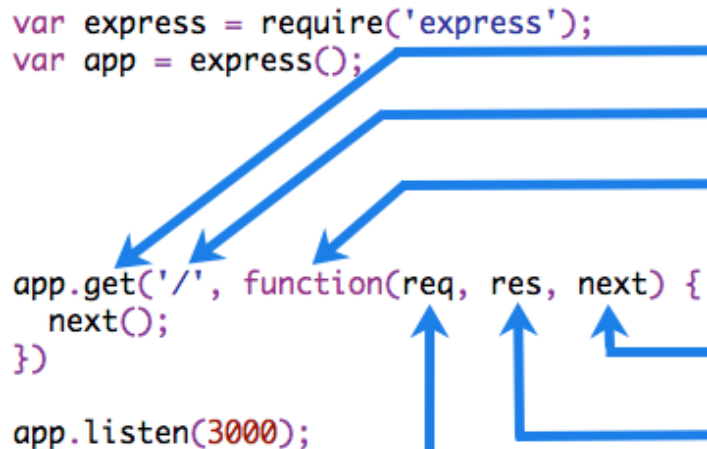
Se mostrará por consola cuando el servidor esté listo para operar y en qué puerto lo está haciendo.

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);
```

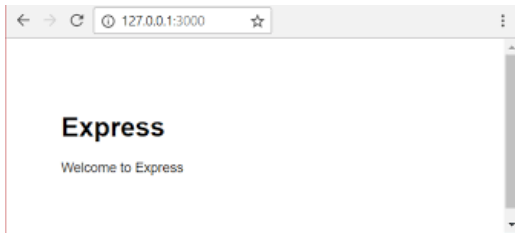


Introducción

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

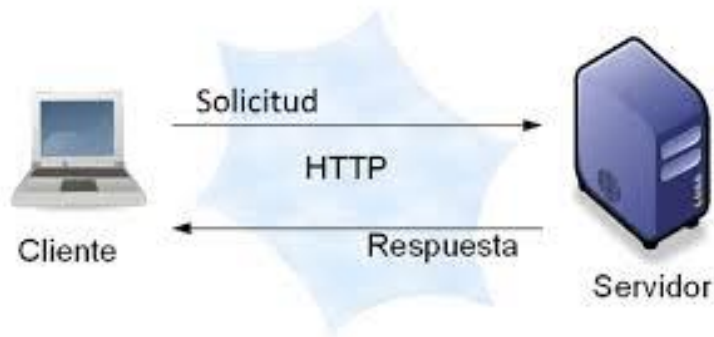
Express como framework soporte para servidores REST



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.*

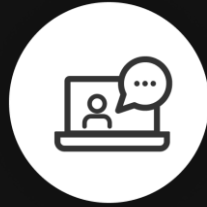
Configuración petición Get

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 raíz del servidor

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

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



***¿Alguna pregunta hasta
ahora?***



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!

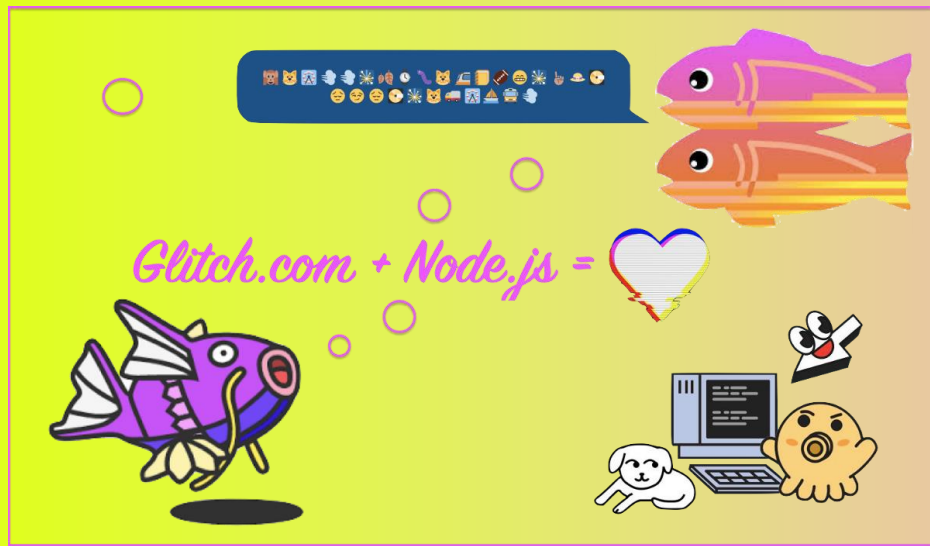
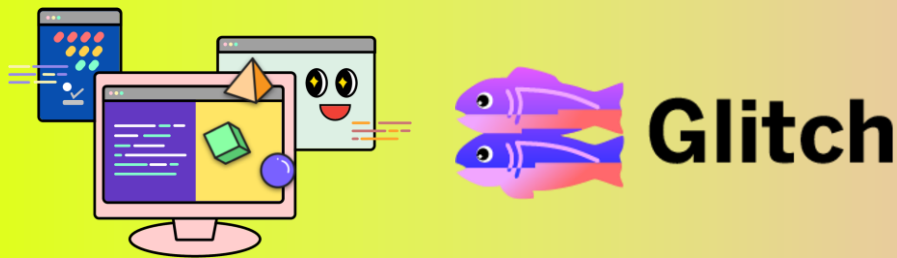


Nuestro primer Despliegue en un servidor en la nube

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



Glitch is now part of Fastly, and we are very excited about what this means for the community! [Read the full announcement here](#)

fastly

Glitch is the friendly place where everyone builds the web. Start a [new blog](#), play with [React](#), or build new worlds with [WebXR](#). Let's go!

**Try Glitch in Bio!**

Our newest starter helps you build your own free links page.

[Remix Glitch In Bio](#)

Manage your projects

You have 1000 active hours per month for all of your projects. [Learn more](#) →

Project Hours

0% | 0/1000 hours/month

Projects you no longer own

Become a Glitch Member

Upgrade your account to give your projects superpowers: Unlimited hours, no rate limits, and boosted specs for up to five projects.

Upgrade

Projects

You own You're a member Archived

Search Projects

☐ Project name

☐ Boosted

☐ Project hours

☐ Last edited



Glitch In Bio

Instantly create a list of your favorite links, and customize them for free.

Remix: [Glitch In Bio](#)

GENERATED STATIC SITE



Website

Build a website from scratch using the basics: HTML, CSS and JavaScript. It could be a splash page.

STATIC SITE





hello-express

Settings

Assets

Files

> public/

> views/

.env

README.md

package.json

server.js

README.md

EDIT MARKDOWN

hello-express

A server that serves a webpage, its resources, and some data

Your Project

On the front-end,

- Edit `views/index.html` to change the content of the webpage
- `public/client.js` is the javascript that runs when you load the webpage
- `public/style.css` is the styles for `views/index.html`
- Drag in `assets`, like images or music, to add them to your project

On the back-end,

- your app starts at `server.js`
- add frameworks and packages in `package.json`
- safely store app secrets in `.env` (nobody can see this but you and people you invite)

Click [Show](#) in the header to see your app live. Updates to your code will instantly deploy.


Made by [Glitch](#)

Glitch is the friendly community where you'll build the app of your dreams. Glitch lets you instantly create, remix, edit, and host an app, bot or site, and you can invite collaborators or helpers to simultaneously edit code with you.

Find out more [about Glitch](#).


(^ x ^)

[Report Abuse](#)[PREVIEW](#)

 Glitch

README.md

RemixShare



express-demo-cloud

SettingsAssetsFiles

> public/
> views/
.env
README.md
package.json
server.js

README.md

EDIT MARKDOWN

hello-express

A server that serves a webpage, its resources, and some data

Your Project

On the front-end,

- Edit `views/index.html` to change the content of the webpage
- `public/client.js` is the javascript that runs when you load the webpage
- `public/style.css` is the styles for `views/index.html`
- Drag in `assets`, like images or music, to add them to your project

On the back-end,

- your app starts at `server.js`
- add frameworks and packages in `package.json`
- safely store app secrets in `.env` (nobody can see this but you and people you invite)

Click [Show](#) in the header to see your app live. Updates to your code will instantly deploy.



Made by [Glitch](#)

Glitch is the friendly community where you'll build the app of your dreams. Glitch lets you instantly create, remix, edit, and host an app, bot or site, and you can invite collaborators or helpers to simultaneously edit code with you.

Find out more [about Glitch](#).

(^ x ^)

STATUSLOGSTERMINTOOLSPREVIEW

<https://glitch.com/>

CODER HOUSE



express-demo-cloud

Settings

Assets

Files

> public/

> views/

.env

README.md

package.json

server.js

server.js

PRETTIER

```
1 const express = require('express')
2
3 const app = express()
4
5 const server = app.listen(process.env.PORT, () => {
6   console.log('Servidor http escuchando en el puerto ${server.address().port}')
7 })
8
9 server.on("error", error => console.log('Error en servidor ${error}'))
10
11 // A) '/' en esta ruta raiz, 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.
12 // 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'
13 // C) '/fyh' donde se devolverá la fecha y hora actual en formato objeto: { fyh: '11/1/2021 11:36:04' }
14
15 app.set('cantidad', 0)
16
17 app.get('/', (req, res) => {
18   res.send('<h1 style="color: blue">Bienvenidos al servidor express</h1>')
19 })
20
21 app.get('/visitas', (req, res) => {
22   let cantidad = app.get('cantidad')
23   cantidad += 1
24   res.send('La cantidad de visitas es ${cantidad}')
25   app.set('cantidad', cantidad)
26 })
27
28 app.get('/fyh', (req, res) => {
29   const dayjs = require('dayjs')
30   res.json({ fyh: dayjs().format('DD/MM/YYYY hh:mm:ss') })
31 })
```



STATUS



LOGS



TERMINAL



TOOLS



PREVIEW



express-demo-cloud

Settings

Assets

Files

> public/

> views/

.env

README.md

package.json

server.js

package.json

PRETTIER

+ ADD PACKAGE

```
1 {
2   "description": "describes your app and its dependencies",
3   "url": "https://docs.npmjs.com/files/package.json",
4   "name": "hello-express",
5   "version": "0.0.1",
6   "description": "A simple Node app built on Express, instantly up and running.",
7   "main": "server.js",
8   "scripts": {
9     "start": "node server.js"
10  },
11  "dependencies": {
12    "express": "^4.17.1",
13    "dayjs": "^1.11.5"
14  },
15  "engines": {
16    "node": "12.x"
17  },
18  "repository": {
19    "url": "https://glitch.com/edit/#!/hello-express"
20  },
21  "license": "MIT",
22  "keywords": [
23    "node",
24    "glitch",
25    "express"
26  ]
27 }
```

Glitch

package.json

Remix Share

express-demo-cloud

Settings

Assets

Files

public/

views/

.env

README.md

package.json

server.js

```
1 {
2   "description": "describes your app and its dependencies",
3   "url": "https://docs.npmjs.com/files/package.json",
4   "name": "hello-express",
5   "version": "0.0.1",
6   "description": "A simple Node app built on Express, instantly up and running.",
7   "main": "server.js",
8   "scripts": {
9     "start": "node server.js"
10  },
11  "dependencies": {
12    "express": "^4.17.1",
13    "dayjs": "^1.11.5"
14  },
15  "engines": {
16    "node": "12.x"
17  },
18  "repository": {
19    "url": "https://glitch.com/edit/#!/hello-express"
20  },
21  "license": "MIT",
22  "keywords": [
23    "node",
24    "glitch",
25    "express"
26  ]
27 }
```

Logs

Clear Debugger

node v12.0.0, with pnpm

Installing ...

Performing headless installation

Total install time: 8605ms

Servidor http escuchando en el puerto 3000

STATUS LOGS TERMINAL TOOLS PREVIEW

<https://glitch.com/>

CODER HOUSE

Bienvenidos al servidor express

<https://glitch.com/>

CODER HOUSE



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.

CODER HOUSE

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
  }
]
```

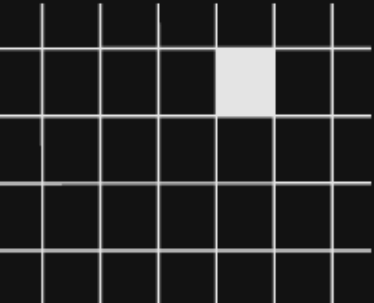
¿PREGUNTAS?





¡MUCHAS GRACIAS!

Resumen de lo visto en clase hoy:

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



OPINA Y VALORA ESTA CLASE

#DEMOCRATIZANDOLAEDUCACIÓN