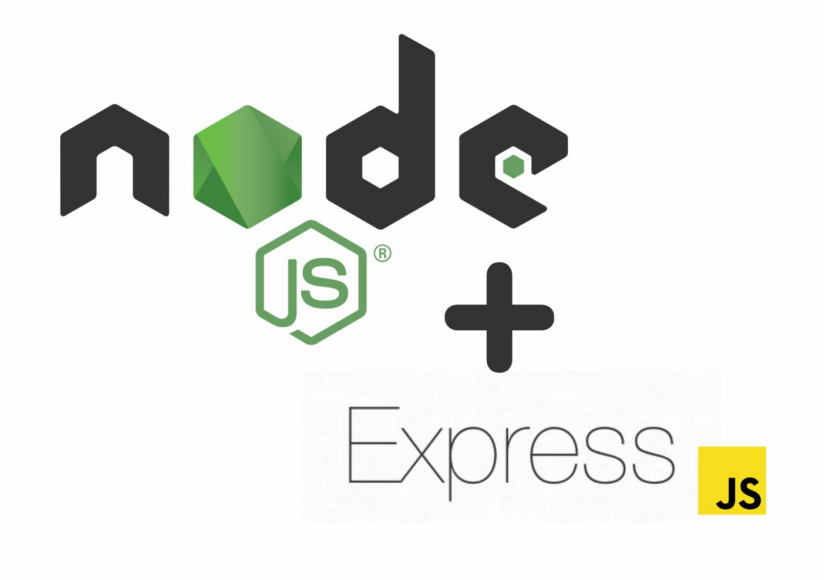
releevant.



linkedin.com/in/maortizolid

maortizolid@gmail.com

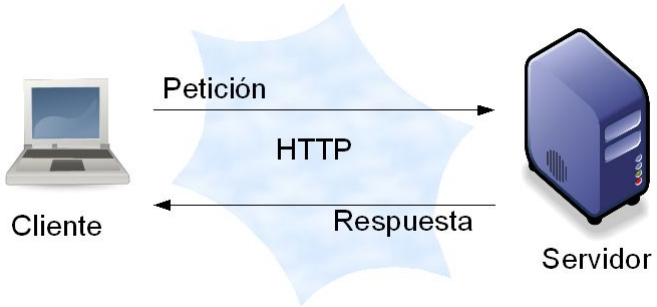
releevant.

Servidores con Node.js y el protocolo HTTP

Según la definición de **HTTP** podríamos afirmar que es un **protocolo de red**. *¿Qué es un protocolo de red?*

> *Un protocolo de red se define como un conjunto de reglas y permisos para establecer la comunicación entre dispositivos. Además se encarga de la configuración de la conexión, es decir cómo y de qué forma se va a actuar.*

HTTP (*Protocolo de Transferencia de Hipertexto*) es el protocolo por excelencia de la web. Se utiliza para todo tipo de transacción, como por ejemplo *que el navegador pida una página o recurso a un servidor y que éste responda.*



releevant.

El módulo http en Node.js. Crear un servidor.

- El módulo **`http`** es uno de los módulos estándar de Node.js que se utiliza para trabajar con el protocolo HTTP. Este módulo proporciona un conjunto de APIs que se pueden utilizar para crear servidores HTTP y clientes HTTP en Node.js.

- Para utilizar el módulo **`http`** en una aplicación de Node.js, primero se debe importar el módulo:

```
const http = require('node:http');
```

- **`http`** es un módulo que ya viene con node, por lo tanto no debemos instalarlo con npm u otro gestor.

releevant.

Crearemos dos variables, una para el `host` y otra para el `puerto`, ya que el servidor necesitará de estos datos.

```
const host = '127.0.0.1';
const port = 3000;
```

Vamos a correr nuestro servidor en nuestra máquina local, por lo tanto con la variable **host** nos vamos a referir a **localhost** usando su dirección IP que es `127.0.0.1`, y por convención, vamos a correr nuestro servidor en el puerto `3000`.

Llamamos al método **createServer** que devuelve una nueva instancia de **http.Server.**

```
const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Primer servidor con Node.Js');
});
```

releevant.

Al método createServer le pasamos un callback con dos parámetros `req` y `res` . ***req*** *es la* ***petición que le hacemos al servidor,*** *en sí, es un objeto que tiene toda la información de la petición, y* ***res*** *de la respuesta. Por lo tanto establecemos el código* ***200*** *(HTTP 200 = exitoso), los* ***headers*** *(cabeceras de contenido) y para finalizar cerramos la conexión con un mensaje.*

Llamamos al método `listen` que concretará la conexión.

```
server.listen(port, host, () => {
  console.log(`Servidor corriendo en http://${host}:${port}`);
});
```

El método **listen** recibe el puerto, el host y un callback, donde le pasamos el mensaje una vez que la conexión se abra.

releevant.

Código completo:

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

const host = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Primer servidor con Node.Js');
});

server.listen(port, host, () => {
  console.log(`Servidor corriendo en http://${host}:${port}`);
});
```

releevant.

También podemos adjudicar cualquier puerto que tengamos disponible pasándole al la función `server.listen` un 0. Esta es una forma conveniente de evitar conflictos de puerto, especialmente si estamos ejecutando varios servidores en la misma máquina y no queremos especificar manualmente un número de puerto.

```
const http = require('node:http');

const host = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Primer servidor con Node.Js');
});

// Adjudicar el primer puerto disponible (0)
server.listen(0, host, () => {
    const port = server.address().port;
    console.log(`Servidor corriendo en http://${host}:${port}`);
})
```

releevant.

El módulo net

El módulo `net` en Node.js proporciona funcionalidades para la creación de servidores TCP (Transmission Control Protocol).

Permite crear servidores que pueden manejar conexiones entrantes a través del protocolo TCP, facilitando la implementación de aplicaciones de red personalizadas.

El método `net.createServer()` se utiliza para crear una instancia de un servidor TCP, y se pueden gestionar eventos en las conexiones entrantes utilizando callbacks.

Vamos a realizar un ejercicio práctico que consiste en la creación de dos archivos en Node.js. Uno de ellos manejará la lógica de búsqueda de un puerto disponible, y el otro levantará un servidor HTTP aplicando la función desarrollada previamente.

Esto nos permitirá entender cómo los módulos net y http pueden combinarse para construir servidores robustos y evitar conflictos de puerto.

releevant.

3.free-port.js

```
// app que nos da un puerto libre
const net = require('node:net');

function findAvailablePort(desiredPort) {
  return new Promise((resolve, reject) => {
    const server = net.createServer()

    server.listen(desiredPort, () => {
      // recuperamos el puerto en caso de que este disponible
      const { port } = server.address()
      // cerramos el servidor y resolvemos la promesa con el puerto
      server.close(() => {
        resolve(port)
      })
    })

    // en caso de que el puerto no este disponible, llamamos a la función de búsqueda de puerto disponible de forma recursiva
    server.on('error', (err) => {
      if (err.code === 'EADDRINUSE') {
        findAvailablePort(0).then(port => resolve(port))
      } else {
        reject(err)
      }
    })
  })
}

module.exports = { findAvailablePort }
```

releevant.

4.call-free-port.js

```
const http = require('node:http');
const { findAvailablePort } = require('./3.free-port.js');

const server = http.createServer((req, res) => {
    res.statusCode = 200;
    res.setHeader('Content-Type', 'text/plain');
    res.end('Primer servidor con Node.Js');
})

// llamamos a la función para asignar un puerto libre si el indicado está ocupado ( findAvailablePort(3000) )
findAvailablePort(3000).then(port => {
    server.listen(port, () => {
        console.log(`listening on port http://localhost:${port}`);
    })
})
```

releevant.

Manejar el módulo `http` de NodeJs y estructurar el código es algo complejo, por eso vamos a utilizar librerías específicas para ello, que nos facilitarán toda esa tarea.

La más extendida es **Express**, que nos permitirá estructurar una Api de manera ágil y nos proporciona funcionalidades como el enrutamiento, además de muchas otras que veremos a continuación.