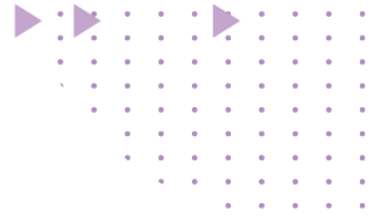


«Talento Tech»

Desarrollo Web 4

Clase 02





Clase N° 2 | Creando un Server

Temario:

- Instalación de Git bash.
- Creación de un servidor web con Node (simple).
- Creación de un servidor con Express.js (1° Parte) y método GET.
- Introducción y testeo con Postman.



Instalación de Git bash



¡Hola, jóvenes programadores! Hoy vamos a explorar Git for Windows, una herramienta imprescindible para trabajar con Git en el mundo de Windows. Prepárense para una emocionante aventura en el reino del control de versiones.

¿Qué es Git for Windows?

Git for Windows es como un hechizo que nos permite utilizar **Git**, esa mágica herramienta de control de versiones, en el entorno familiar de Windows. Es como tener un amigo confiable que nos ayuda a rastrear los cambios en nuestro código y colaborar con otros.

¿Por qué usar Git for Windows?

Imaginen que están construyendo un castillo de código y, de repente, quieren volver en el tiempo para ver cómo era el castillo hace un mes. ¡Eso es lo que hace Git! Y con Git for Windows, pueden hacerlo de manera fácil y amigable.

Instalación de Git for Windows

Vamos a sumergirnos en el proceso de instalación. Siganme en estos pasos simples:

1. Descarga:

Vayan al sitio web oficial de Git for Windows "[Git for Windows](#)"
Descarguen el instalador.

2. Ejecución del instalador:

Abren el instalador que descargaron. Este será nuestro mapa para entrar en el mundo Git.

3. Configuración:

Sigamos las indicaciones del instalador. Pueden elegir las opciones predeterminadas si no están seguros.

4. Selección del editor:

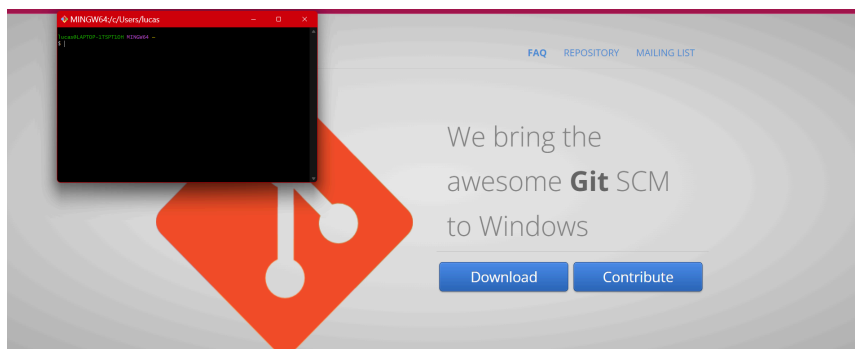
Elijan su editor de texto favorito. Este será su lápiz para escribir las historias de su código.

5. Configuración adicional:

Ajusten las opciones según sus preferencias. Pueden ser aventureros o seguir el camino más transitado.

6. Finalización:

Si llegaste hasta esta ventana, realizaste exitosamente la instalación :)





¡Hemos instalado Git for Windows! Ahora, abran **Git Bash** desde el menú de inicio o hagan clic derecho en una carpeta y seleccionen "Git Bash Here". Esta será su varita mágica para interactuar con Git.

🚀 ¡Practiquemos un poco!

Escribí este comando para ver la versión de Git que instalamos:

```
git --version
```

¡Bravo! Han dado su primer paso en el fascinante mundo de Git for Windows.

En nuestras próximas clases, exploraremos cómo usar Git para controlar versiones y colaborar en proyectos mágicos. ¡Hasta la próxima, jóvenes programadores! 🚀 ✨

Creación de un servidor web con Node(simple)

Para acceder a las páginas web de cualquier aplicación web, necesitaremos un servidor. El servidor web maneja todas las solicitudes http de la aplicación, por ejemplo "**Apache**" es un servidor web, tanto como para **PHP o Java**.

Ahora veamos un ejemplo de cómo crear y correr un servidor. Nuestra aplicación creará un servidor web que escuchará al puerto 3000. Si realizamos una solicitud a través del navegador a este puerto, la aplicación nos enviará una respuesta.

Vamos a crear un archivo llamado servidor.js con el siguiente código:

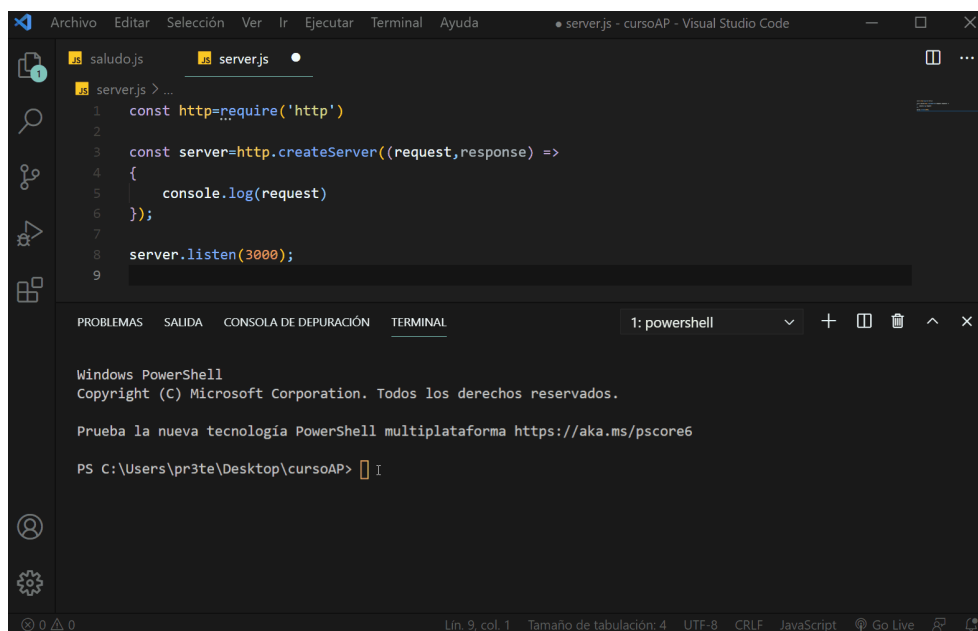
```
4 const http=require('http')
5 const server=http.createServer( (request,response) =>
6   {
7     console.log(request);
8   });
9 server.listen(3000);
```

Inspeccionamos cómo funciona este código:

```
4 const http=require("http") 1
5 const server=http.createServer( (request,response) => 2
6   {
7     console.log(request); 3
8   });
9   server.listen(3000); 4
```

1. La labor básica de la función “require” es que, lee un archivo **JavaScript**, ejecuta el archivo y luego procede a devolver el objeto de exportación. Entonces, en nuestro caso, dado que queremos usar la funcionalidad del módulo `http`, usamos la función “require” para obtener las funciones deseadas del módulo `http` para que pueda usarse en nuestra aplicación.
2. En esta línea de código, estamos creando nuestro servidor y como podemos ver se basa en una función simple. Esta función se llama cada vez que se realiza una solicitud a nuestro servidor.
3. Cuando nuestro servidor reciba una solicitud, se enviará y mostrará por consola el objeto “request”.
4. Entonces usamos la función “**server.listen**” para hacer que nuestra aplicación de servidor escuche las solicitudes del cliente en el puerto no 3000. Puede especificar cualquier puerto disponible aquí.

¡Listo ya tenemos el código para nuestro primer servidor! Ahora lo que tenemos que hacer para poder correrlo, es ejecutar nuestro archivo “**server.js**” desde la terminal



```

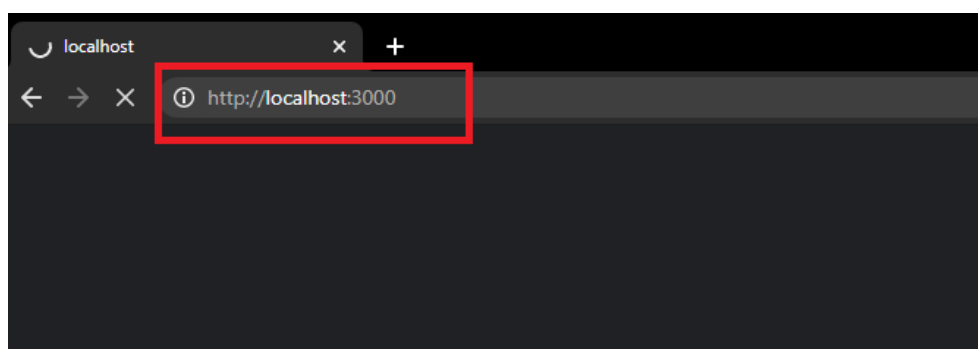
1  const http=require('http')
2
3  const server=http.createServer((request,response) =>
4  {
5      console.log(request)
6  });
7
8  server.listen(3000);
9
    
```

Windows PowerShell
 Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma <https://aka.ms/pscore6>

PS C:\Users\pr3te\Desktop\cursoAP>

Para poder ver el resultado tendremos que navegar hacia la url **localhost:3000**



Lo que nos llegará a nuestra consola es el objeto “**request**”.

Por el momento nuestro servidor cuando recibe una solicitud no envía ninguna respuesta. Más adelante veremos cómo enviar respuestas desde nuestro servidor.



Ahora vamos a hacer uso de **ES Module** para ahorrarnos posibles errores al usar “require”. Recordemos que el “**require**” es usado de manera predeterminada, en este curso elegimos *ES Module*.

Integrar módulos de **ECMAScript** (*ES modules*) en **Node.js** es bastante sencillo. Aquí te dejo una guía paso a paso para que puedas usar “**import**” y “**export**” en lugar de “**require**” en tu servidor web Node.js:

Paso 1: Verificar la Versión de Node.js

Asegúrate de tener una versión de Node.js que sea compatible con *ES modules*. A partir de **Node.js 13**, los módulos ES son compatibles sin la necesidad de la bandera **--experimental-modules**. Sin embargo, si estás utilizando una versión anterior, puedes usar la bandera **--experimental-modules**.

Paso 2 (opcional): Estructura de Carpetas

Asegúrate de tener una estructura de carpetas y archivos adecuada para trabajar con módulos **ES**.
Por ejemplo:

```
- proyecto/  
  - src /  
    - main.js  
  - package.json
```

Paso 3: Configurar el package.json

Asegúrate de tener un archivo package.json en la raíz de tu proyecto. Dentro del archivo, agrega la siguiente configuración:


```
{
  "type": "module",
  "scripts": {
    "start": "node src/main.js"
  }
}
```

El campo **"type": "module"** indica a **Node.js** que debe interpretar los archivos como módulos ES. También configuramos un script de inicio para ejecutar nuestro archivo principal.

Paso 4: Crear un Módulo ES

Dentro de la carpeta "src", crea un archivo llamado "main.js" (o el nombre que desees). Este será tu archivo principal y puede contener código como este:

```
// main.js
import http from 'http';
const servidor = http.createServer((req, res) => {
  res.writeHead(200, { 'Content-type': 'text/plain' });
  res.end('¡Hola, mundo!');
});

const puerto = 3000;
servidor.listen(puerto, () => {
  console.log(` Servidor en funcionamiento en el puerto ${puerto} `);
});
```

En este ejemplo, estamos utilizando import para cargar el módulo http.

CIS

CommonJS

```
const elem = require('module');

module.exports = {
};
```



ESM

ES Modules

```
import { elem } from './module.js';

export const elem = {
};
```



LenguajeJS.com

Paso 5: Ejecutar el Servidor

Abre tu terminal, navega hasta la carpeta de tu proyecto y ejecuta:

```
npm start
```

Esto iniciará tu servidor web **Node.js**. Si todo está configurado correctamente, verás el mensaje "Servidor en funcionamiento en el puerto 3000" en la consola.

¡Listo! Ahora estás utilizando módulos ES en tu servidor web Node.js. Este enfoque te permite aprovechar las características más modernas y limpias de **ECMAScript**. ¡Buena programación!



Creación de un servidor con Express.js (1° Parte) y método GET.

Introducción a Express.js

Express.js es un marco (framework) web para **Node.js** que simplifica el desarrollo de aplicaciones web y APIs. Se centra en facilitar la creación de rutas, el manejo de solicitudes y respuestas, y la integración de middleware para agregar funcionalidades a tu aplicación.

¿Para qué sirve Express.js?

Express.js es utilizado para construir aplicaciones web y APIs de manera rápida y eficiente. Algunas de las cosas que puedes hacer con Express incluyen:

Manejo de Rutas: Define rutas para diferentes URL en tu aplicación y especifica cómo manejar las solicitudes y generar las respuestas.

Middleware: Utiliza middleware para agregar funciones adicionales a tu aplicación, como autenticación, compresión de respuesta, manejo de errores, entre otros.

Servir Archivos Estáticos: Configura Express para servir archivos estáticos como HTML, CSS, y archivos de imágenes.

Integración con Bases de Datos: Puedes conectar Express con bases de datos como MongoDB, MySQL, u otros para almacenar y recuperar datos.

Desarrollo de APIs RESTful: Express es ideal para construir APIs RESTful, permitiéndote crear servicios web que pueden ser consumidos por otras aplicaciones.

API RESTful: ¿Cómo lo vería en un ejemplo?

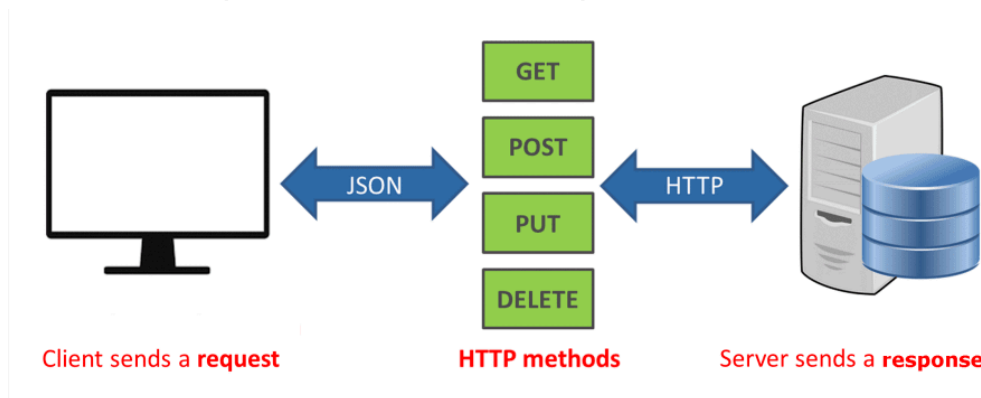
Imagina que tienes un conjunto de datos relacionados con libros en una biblioteca. Puedes diseñar una API RESTful para interactuar con estos datos.

👉 Aquí hay una representación simple utilizando un array de objetos:

Recursos: Cada libro se considera un recurso, y cada libro tiene sus propios atributos, como título, autor y año de publicación. Cada libro se identifica por un ID único.

```
const puerto = [
  { id: 1, titulo "El Señor de los Anillos", autor: "J.R.R. Tolkien", anioPublicacion: 1954},
  { id: 2, titulo: "Cien años de soledad", autor: "Gabriel García Márquez", anioPublicacion: 1967},
  //... otros libros ...
];
```

Operaciones HTTP (Get, Post, Put, Delete)



Con HTTP pueden realizar las siguientes operaciones:

GET: Para obtener información sobre un libro o todos los libros.

```
//Ejemplo de solicitud GET para obtener todos los libros
app.get('/libros', (req, res) => {
  res.json(libros);
});
```

POST: Para agregar un nuevo libro

```
//Ejemplo de solicitud POST para agregar un nuevo libro
app.post ('/libros', (req, res) => {
  const nuevoLibro = req.body;
  libros.push (nuevoLibro);
  res.json(nuevoLibro);
});
```

PUT: Para actualizar la información de un libro existente.

```
//Ejemplo de solicitud PUT para actualizar un libro existente
app.put ('/libros', (req, res) => {
  const libroID = req.params.id;
  // Lógica para actualizar el libro con el ID proporcionado
  res.send( 'Libro actualizado');
});
```

DELETE: Para eliminar un libro.

```
//Ejemplo de solicitud DELETE para eliminar un libro
app.delete ('/libros/id', (req, res) => {
  const libroID = req.params.id;
  // Lógica para eliminar el libro con el ID proporcionado
  res.send( 'Libro eliminado');
});
```



Representación de Recursos

Puedes enviar o recibir datos en formato JSON para representar los libros.

```
// Ejemplo de representación JSON de un libro
{
  "id": 1,
  "titulo": "El señor de los anillos",
  "autor": "J.R.R. Tolkien",
  "anioPublicacion": 1956
}
```

Estado de la Aplicación

La información sobre libros y cualquier acción relacionada está contenida en cada solicitud. Por ejemplo, al enviar una solicitud PUT para actualizar un libro, la solicitud incluirá todos los datos actualizados necesarios.

Sin Estado

Cada solicitud es independiente. La API no almacena información del estado de las solicitudes anteriores.

HATEOAS

Aunque no se muestra en el ejemplo, podrías incluir enlaces o rutas dentro de las respuestas para guiar al cliente sobre qué operaciones puede realizar a continuación.

👉 En resumen, esta representación utilizando un array de objetos ilustra cómo los principios de una **API RESTful** pueden aplicarse a un **conjunto de datos simple**, como una colección de libros en una biblioteca.



Ejemplo Sencillo de Código con Express.js

Vamos a crear un pequeño servidor web usando Express. Este ejemplo responderá con un mensaje simple cuando accedas a la ruta principal.

Paso 1: Instalar Express

Asegúrate de tener Node.js instalado y crea un proyecto de Node.js usando npm init. Luego, instala Express con el siguiente comando:

```
npm install express
```

Paso 2: Crear el Código de Express

En tu archivo main.js (o como lo hayas llamado), agrega el siguiente código:

```
// main.js
import express from 'express';
const app = express ();
const puerto = 3000;

//Ruta principal
app.get('/', (req, res) => {
  res.send ('Hola, mundo con Express!');
});

//Iniciar el servidor
app.listen(puerto, () => {
  console.log('Servidor en funcionamiento en el puerto ${puerto}');
});
```




Paso 3: Ejecutar el Servidor

Guarda los cambios y ejecuta tu aplicación con:

```
npm main.js
```

Paso 4:

Visita <http://localhost:3000> en tu navegador y deberías ver el mensaje "¡Hola, mundo con Express!".

👉 Este es un ejemplo básico, pero muestra cómo puedes crear un servidor web con una ruta usando Express. A medida que te familiarices con Express, podrás construir aplicaciones web más complejas y poderosas. ¡Éxito con tu desarrollo!

Método GET

Ahora vamos a conocer el método GET como introducción a los distintos verbos que podemos usar.

El método GET en HTTP se utiliza para solicitar datos de un servidor. Es seguro y no tiene efectos secundarios. Se realiza a través de la URL y generalmente se usa para obtener información de un recurso específico. En Express.js, puedes manejar solicitudes GET con el método `get` de la aplicación.

Este código es una implementación básica de un servidor web utilizando Express.js. Aquí hay una breve explicación de cada parte:

Importación de Módulos:

```
const express = require('express')
const bodyParser = require('body-parser')
```



Se importan los módulos express y body-parser. express es el marco web que facilita la creación de servidores en Node.js, y body-parser es un middleware de Express que ayuda a analizar los datos del cuerpo de las solicitudes HTTP, especialmente útil para datos en formato JSON.

Creación de la Aplicación Express:

```
const app = express();  
const port = 3000;
```

Se crea una instancia de la aplicación Express y se establece el número de puerto en el que el servidor escuchará las solicitudes.

Middleware Body Parser:

```
app.use(bodyParser.json());
```

Se utiliza el middleware body-parser para analizar el cuerpo de las solicitudes como datos JSON. Esto es esencial cuando se manejan solicitudes que contienen datos en formato JSON.

Datos Iniciales - Variable Global:

```
let data = [  
  { id: 1, name: 'Objeto 1'},  
  { id: 2, name: 'Objeto 2'},  
  
  //Puedes agregar más objetos según sea necesario  
];
```

Se define una variable global data que contiene un array de objetos. Este array simula una fuente de datos que podría provenir de una base de datos o cualquier otra fuente.

Ruta GET para Obtener Objetos:

```
app.get('/api/object') , (req, res) => {  
  res.json(data);  
});
```



Se define una ruta GET (/api/objects) que devuelve los datos almacenados en la variable data como respuesta JSON. Esta ruta simula la obtención de objetos de una API.
Iniciar el Servidor:

```
app.listen(port, () => {  
  console.log(`Servidor Escuchando http://localhost:${port}`);  
});
```

Se inicia el servidor, y se imprime un mensaje en la consola indicando la dirección en la que se está escuchando.

En resumen, este código establece un servidor básico con Express, utiliza body-parser para analizar datos JSON y proporciona una ruta GET que devuelve un array de objetos. Puedes expandir este código según tus necesidades específicas.

Introducción y testeo con Postman.

¿Qué es Postman?

Postman es una herramienta colaborativa de desarrollo de API que simplifica la creación, prueba y documentación de APIs. Es especialmente útil para probar solicitudes HTTP y obtener respuestas fácilmente. Postman ofrece una interfaz gráfica fácil de usar para trabajar con APIs, lo que lo convierte en una herramienta esencial para desarrolladores y equipos de desarrollo.

Principales Características de Postman

Creación de Solicitudes HTTP:

Permite crear y enviar solicitudes HTTP como GET, POST, PUT, DELETE, etc.



Entorno de Desarrollo:

Facilita la organización de variables de entorno y valores para probar diferentes configuraciones.

Automatización de Pruebas:

Ofrece la capacidad de escribir scripts de prueba y automatizar el proceso de prueba.

Colecciones de Solicitudes:

Permite agrupar solicitudes relacionadas en colecciones para una gestión fácil y compartirlas con otros.

Documentación Automática:

Genera automáticamente documentación a partir de las solicitudes y respuestas, facilitando la comprensión de la API.

¿Cómo usar Postman?

Para usar Postman con el código proporcionado y realizar pruebas de las rutas definidas, sigue estos pasos:

1. Iniciar el Servidor:

Asegúrate de que tu servidor Express esté en ejecución. Si no lo has iniciado, ejecuta tu archivo de código en la terminal con el comando:

```
node tu-archivo.js
```

Verifica que el servidor está escuchando en el puerto 3000.

2. Abrir Postman:

Abre la aplicación de Postman en tu computadora.

3. Crear una Nueva Solicitud:



Haz clic en el botón **"New"** para crear una nueva solicitud.

4. Configurar la Solicitud GET:

En la nueva ventana, selecciona el método GET.

Ingresa la URL de tu servidor, por ejemplo: <http://localhost:3000/api/objects>.

5. Enviar la Solicitud:

Haz clic en el botón "Send" para enviar la solicitud GET.

6. Ver la Respuesta:

Deberías ver la respuesta de tu servidor, que en este caso será un array de objetos JSON.

👉 Estos pasos te permitirán realizar pruebas básicas de las rutas definidas en tu servidor Express utilizando Postman. Puedes explorar diferentes tipos de solicitudes, enviar datos en el cuerpo de las solicitudes y verificar las respuestas del servidor. Esto es especialmente útil para validar el comportamiento de tus rutas antes de integrarlas en tu aplicación. A lo largo del curso iremos conociendo otras solicitudes.



Desafío #1 (Obligatorio)

En este primer desafío vamos a crear un servidor en **express.js** que nos permita enviar por un método get un “Hola Mundo”.

Recordá tener inicializado el proyecto con **npm init -y**, tener instalado express con el comando de consola **npm i express** para poder llevarlo a cabo.

```
import express from 'express';

const app = express();
const puerto = 3000;

//Ruta Principal
app.get('/', (req, res) => {
  res.send('¡Hola, mundo con Express! ');
});

//Iniciar el servidor
app.listen(puerto, () => {
  console.log(`Servidor en funcionamiento en el puerto${puerto}`);
});
```



Buenos Aires
aprende
Agencia de Actividades para el Futuro

BA Buenos
Aires
Ciudad