

# Desarrollo Backend

Bienvenid@s

Servidores Web I

Clase 04



Pon a grabar la clase



# Temario

- Inicializar un proyecto Node.js
  - Comandos básicos de node
  - El archivo package.js
  - Configuración básica
- Servidores Web
  - Qué son
  - Cómo se piensa su lógica
- El módulo HTTP
  - Declaración
  - Métodos
  - Creación de un servidor básico
  - Probar su funcionamiento



# Inicializar un proyecto Node.js

**Finalizamos el repaso general de los conceptos más importantes del lenguaje JS, como para estar a tono en lo que respecta a la creación de proyectos backend.**



**Vamos entonces a adentrarnos en el uso intensivo de Node.js y JavaScript, para sacar el máximo provecho de todas sus características.**

# Inicializar un proyecto Node.js

# Inicializar un proyecto Node.js

## ¿Cómo inicializar un proyecto en Node.js?

Si bien podemos ejecutar archivos JS desde una ventana Terminal utilizando el comando **node**, cada proyecto en Node.js requiere de una estructura básica que debemos tener presente.

Esto nos facilitará luego, poder instalar dependencias, frameworks, librerías y/o agregar archivos/recursos externos.



# Inicializar un proyecto Node.js

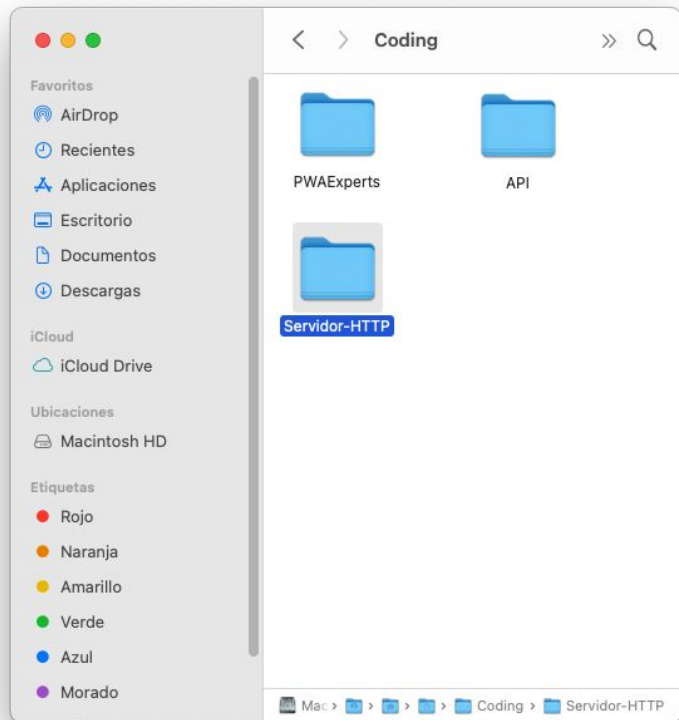
**Para iniciar un proyecto en Node.js, hay algunos pocos pasos iniciales que debes seguir. Veamos cuáles son, a continuación:**

1. **Instalar Node.js** ✓
2. Crear una carpeta para tu proyecto
3. Inicializar el proyecto con npm
4. Verificar/Configurar el archivo package.json
5. Crear un archivo de entrada



Lo trabajaremos como una receta, para no olvidar ninguno de los ingredientes.

# Inicializar un proyecto Node.js



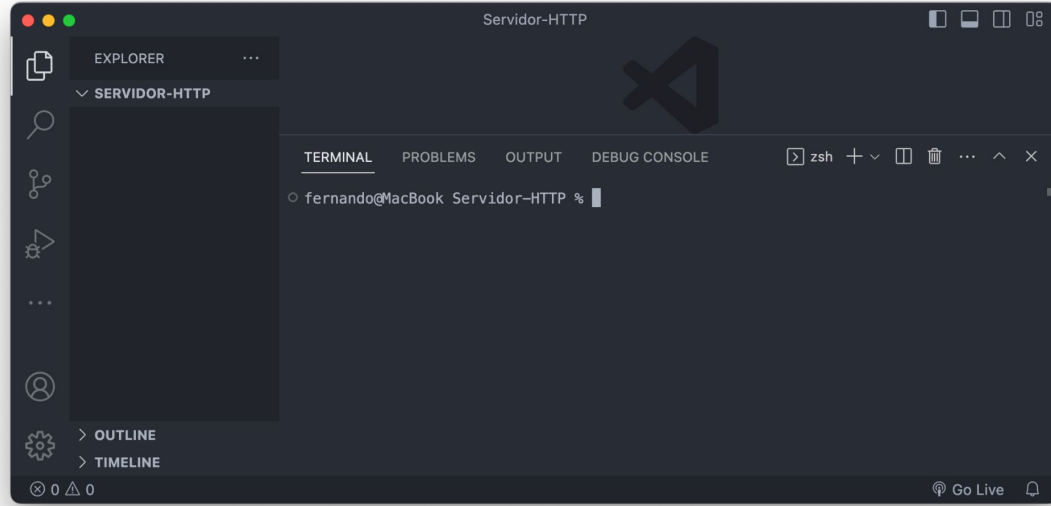
Ya tenemos resuelto el editor de código (**VS Code**), y la instalación y prueba de funcionamiento de **Node.js**. Por lo tanto, el paso que nos corresponde es crear la carpeta que contendrá nuestra aplicación a desarrollar.

Define un espacio fijo para dicha carpeta, pensando en que harás un backup de la misma en un repositorio como **Github**.

Para alinear este proceso con el tema a tratar hoy, crea una carpeta llamada **Servidor-HTTP**.



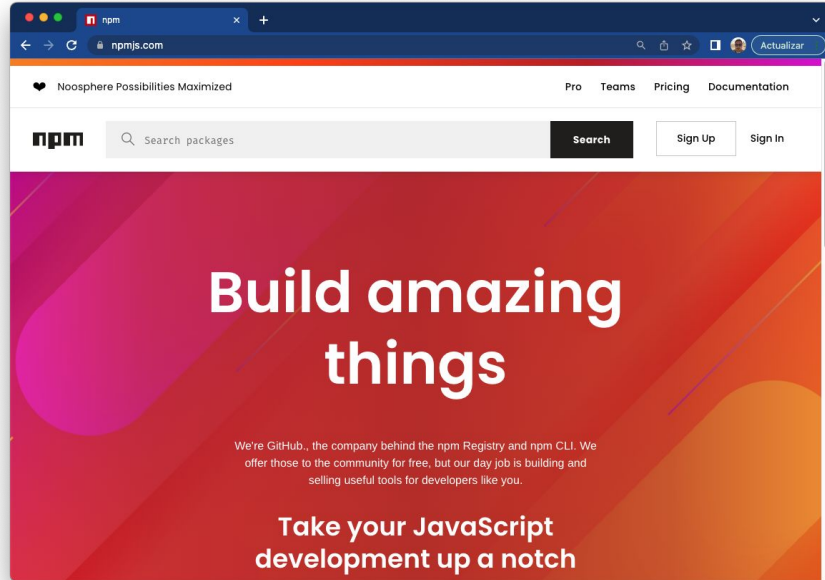
# Inicializar un proyecto Node.js



Desde VS Code, con la carpeta de proyecto abierta, visualizamos la consola o ventana Terminal. En ella, utilizaremos una herramienta denominada NPM para poder crear los pilares de todo proyecto **Node.js**.

# NPM

# NPM



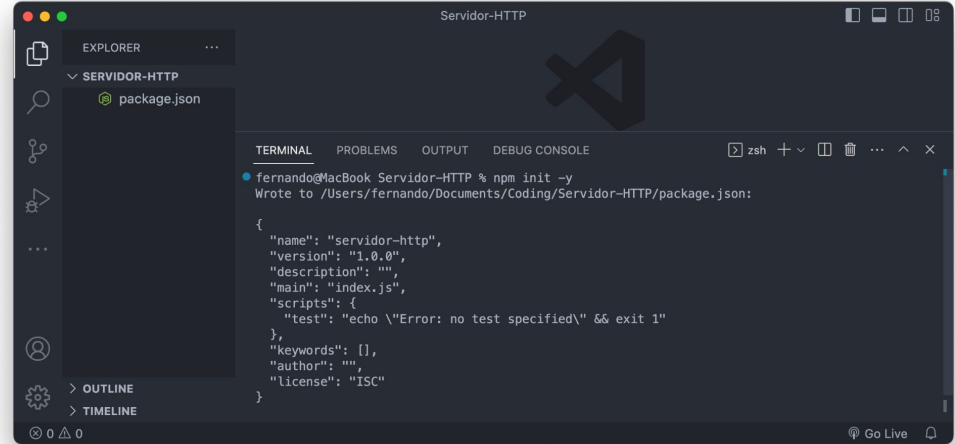
**NPM** es un **gestor de dependencias** para **Node.js** el cual **permite**, entre otras cosas, **inicializar un proyecto Node**, e **instalar dependencias** en el mismo.

La herramienta NPM la utilizamos por línea de comando. Por otro lado, NPM cuenta con un sitio web dedicado, donde almacena información sobre los frameworks y librerías a instalar en nuestros proyectos, versiones, etc.

# NPM

**npm init** es el nombre del comando que debemos escribir en la Terminal, seguido de un **Enter**. Esta acción nos pedirá un montón de información, que aún no conocemos.

Para evitarlo, ejecutemos en su lugar el comando **npm init -y**. El proceso de configuración inicial será automático.

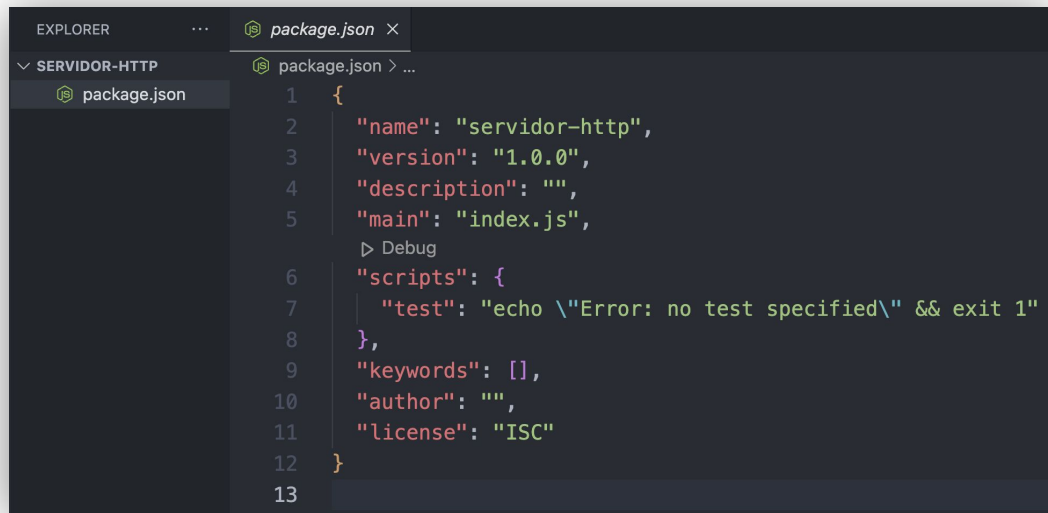
A screenshot of a Visual Studio Code editor window titled 'Servidor-HTTP'. The Explorer sidebar on the left shows a folder named 'SERVIDOR-HTTP' containing a file named 'package.json'. The main editor area displays the content of 'package.json' after running the command 'npm init -y'. The terminal at the bottom shows the command execution and the resulting JSON structure.

```
fernando@MacBook Servidor-HTTP % npm init -y
Wrote to /Users/fernando/Documents/Coding/Servidor-HTTP/package.json:

{
  "name": "servidor-http",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

# NPM

**NPM** crea un archivo llamado **package.json**. Este archivo almacena la configuración inicial de nuestro proyecto node, y **define bajo una estructura de pares clave-valor**, la configuración básica del mismo (*nombre, versión, archivo principal, etc.*)



```
1  {
2    "name": "servidor-http",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "scripts": {
7      "test": "echo \\\"Error: no test specified\\\" && exit 1"
8    },
9    "keywords": [],
10   "author": "",
11   "license": "ISC"
12 }
```

Además, en este archivo, se irá registrando cada nueva dependencia que agreguemos a nuestro proyecto, a medida que este crezca.

# NPM

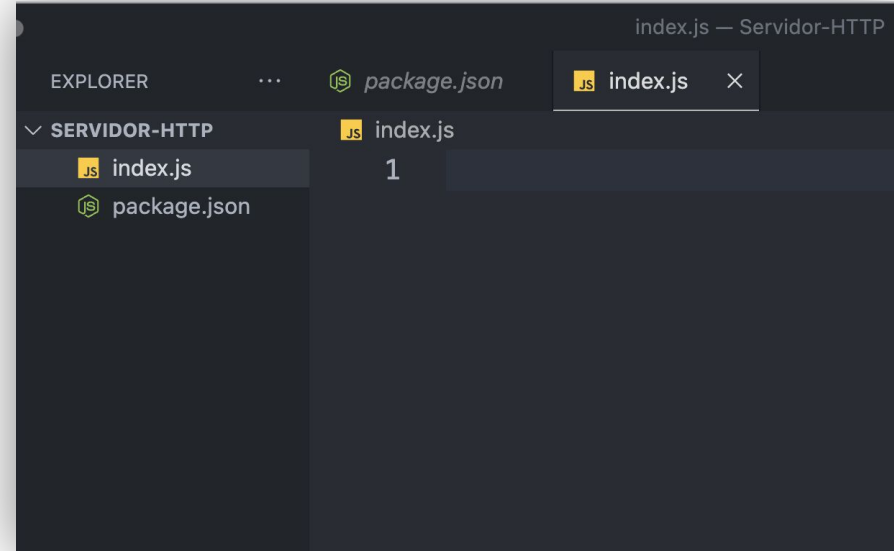
**NPM es una herramienta mucho más poderosa, y ofrece muchas otras opciones que facilitan nuestra tarea como programadores. Más adelante seguiremos inspeccionando todo el poder de NPM como también del archivo Package.json.**

# NPM






Nos queda crear el archivo principal de nuestro proyecto. **Node.js** sugiere el nombre **index.js** para el mismo.

Podemos optar por respetarlo, o crear un archivo con cualquier otro nombre.

Si hacemos esto último, procuremos realizar el cambio en el archivo **package.json**, de **index.js** por el nombre de archivo elegido por nosotros.



# Inicializar un proyecto Node.js

1. **Instalar Node.js** 
2. **Crear una carpeta para tu proyecto** 
3. **Inicializar el proyecto con npm** 
4. **Verificar/Configurar el archivo package.json** 
5. **Crear un archivo de entrada** 

Así de rápido completamos los pasos de nuestra receta. Con esto, ya tenemos creada la base de cada proyecto Node.js que deseemos realizar.

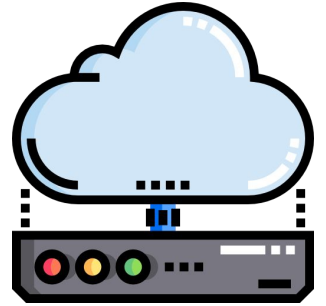


# Servidores Web

# Servidores Web

**Los servidores web son programas o sistemas** informáticos **que proveen contenido y servicios web**, usualmente, a través de Internet.

Un servidor web **recibe y procesa las solicitudes** de los usuarios, y **luego responde con los datos solicitados**, como páginas web, imágenes, videos y otros archivos.



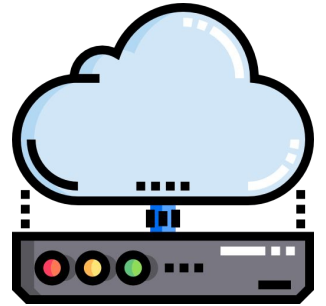
**UNTREF**

UNIVERSIDAD NACIONAL  
DE TRES DE FEBRERO

# Servidores Web

También, alojan y **sirven archivos y datos a través de un protocolo de transferencia de hipertexto (HTTP)** o su versión segura (HTTPS).

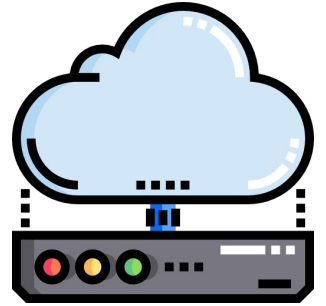
Los sitios web se crean y se cargan en un servidor web, y los usuarios acceden a ellos mediante un navegador web, que solicita estos archivos al servidor, para mostrarlos en el browser.



# Servidores Web

Algunos de los **servidores web más populares** son [Apache](#), [Nginx](#), [IIS](#) (*Internet Information Services*) y [LiteSpeed](#).

Estos servidores web se utilizan para alojar sitios web de diferentes tamaños y complejidades, desde pequeñas páginas personales hasta grandes portales de comercio electrónico o sitios web de medios de comunicación.



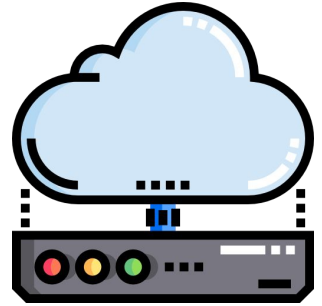
**UNTREF**

UNIVERSIDAD NACIONAL  
DE TRES DE FEBRERO

# Servidores Web

En una aplicación de backend, el **servidor web** es **responsable de recibir y responder a las solicitudes de los clientes**, y de coordinar la interacción entre los diferentes componentes de la aplicación.

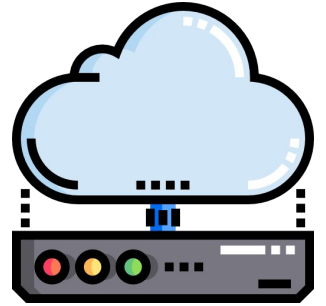
Estas solicitudes son recibidas a través de HTTP y las enruta a la aplicación adecuada para su procesamiento.



# Servidores Web

También puede ser responsable de **autenticar y autorizar usuarios**, **manejar la seguridad de la aplicación** y la **encriptación de datos**, y gestionar las **sesiones de usuarios**.

Además, puede ser utilizado para **acelerar el rendimiento** de la aplicación, manejar los **resultados de base de datos** u otras fuentes, **distribuir las solicitudes entrantes** a otros servidores, para **balancear la carga de uso** cuando hablamos de aplicaciones backend de grandes empresas.



# Servidores Web

**En resumen, el servidor web es una parte esencial de las aplicaciones de backend, ya que proporciona la infraestructura necesaria para que la aplicación se comuniquen con los clientes y gestione la complejidad de la interacción de la aplicación.**

# Servidores Web

¿Y en Node.js, qué papel cumple un servidor web? 🤔

En nuestro caso, no necesitamos un servidor web como los mencionados anteriormente. Contamos con las herramientas necesarias para crear uno de acuerdo a nuestra necesidad.

**Para lograrlo, combinaremos a Node.js con el lenguaje JavaScript, y así, crear nuestro propio webserver.**



**UNTREF**

UNIVERSIDAD NACIONAL  
DE TRES DE FEBRERO



# Servidores Web

```

Servidor Web

//PÚBLICO (DISPONIBLE A TRAVÉS DE INTERNET)
https://mi-aplicacion-backend.com/

//PRIVADO (DENTRO DE UNA RED CORPORATIVA)
https://servidor-web-backend/app

https://192.168.156.202/app

//LOCAL (EN NUESTRA COMPUTADORA / AMBIENTE DE DESARROLLO)
http://localhost:4000/

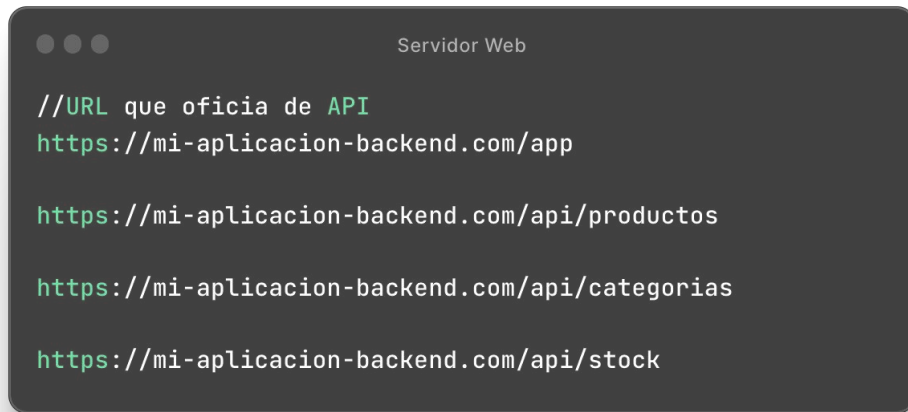
http://127.0.0.1:4000/

```

En estos ejemplos encontramos una representación de diferentes formas en que se puede “ver”, a un servidor web, una vez que éste se encuentra activo.

Cada ambiente de trabajo del servidor web, lo hace accesible a través de una URL que cambia ligeramente.

# Servidores Web



```

//URL que oficia de API
https://mi-aplicacion-backend.com/app

https://mi-aplicacion-backend.com/api/productos

https://mi-aplicacion-backend.com/api/categorias

https://mi-aplicacion-backend.com/api/stock

```

Como vimos en el ejemplo anterior, **un servidor web es accesible a través de una URL**. La URL será la encargada, cuando creamos nuestra aplicación backend, de ser el medio de comunicación entre las aplicaciones frontend, y los datos que nuestra aplicación backend le proveerá a éstas.

# Servidores Web

Servidor Web

`https://mi-aplicacion-backend.com/api/productos/`

URL base

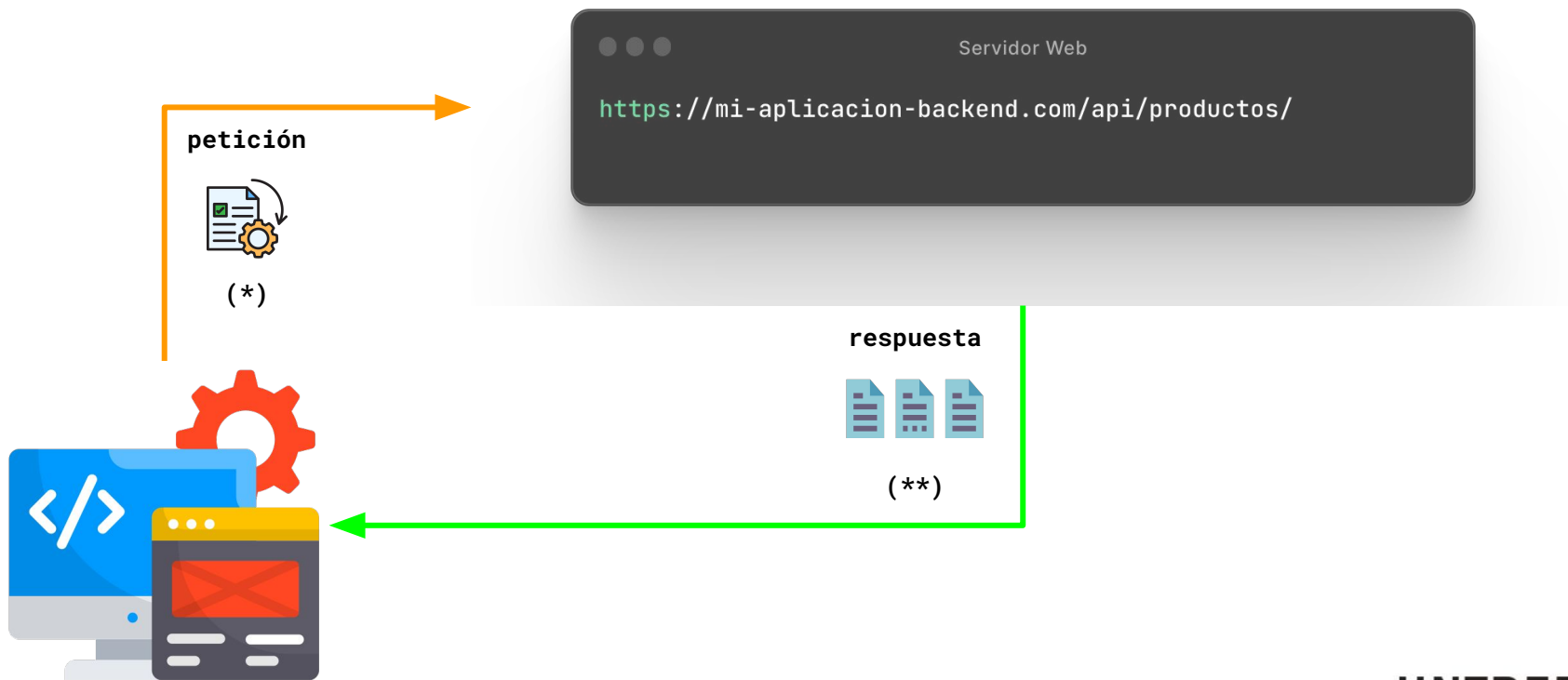
Ruta que aloja la aplicación backend

Servicio/instrucción/Recurso en nuestra aplicación backend

**UNTREF**

UNIVERSIDAD NACIONAL  
DE TRES DE FEBRERO

# Servidores Web



# Servidores Web

**¿Y entonces, cómo creo un servidor web? 🤔**

Para nuestro primer proyecto, definiremos una aplicación Node.js, la cual al ejecutarse, inicializará un servidor web.

El mismo estará disponible bajo la URL <http://localhost:3000/> o similar. Y podremos probar su funcionamiento, desde un navegador web, accediendo a esta URL, y todo desde nuestra misma computadora.

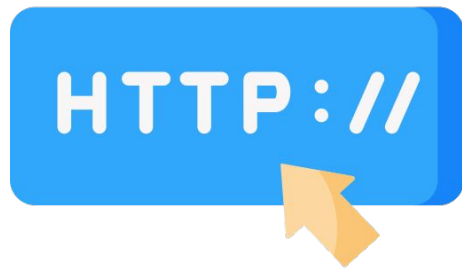


# El módulo HTTP

# El módulo HTTP

Como vimos en toda la etapa introductoria de la cursada, **HTTP** hace referencia a un protocolo el cual nos permite, entre otras cosas, navegar por Internet.

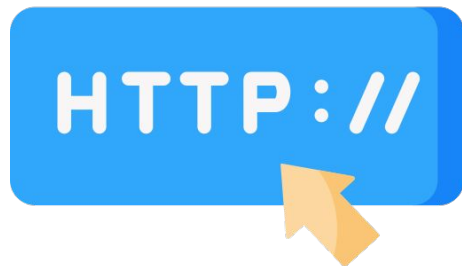
Todos los sitios web a los cuales accedemos como usuarios, utilizan el término **HTTP** ó **HTTPS** delante del resto de la dirección web (URL).



# El módulo HTTP

El entorno de ejecución de Node.js, cuenta con un módulo interno (*el cual forma parte del Core, o núcleo, de Node.js*), llamado **módulo u objeto HTTP**.

Este será nuestro aliado para que podamos crear nuestro primer servidor web, y ponerlo en funcionamiento.

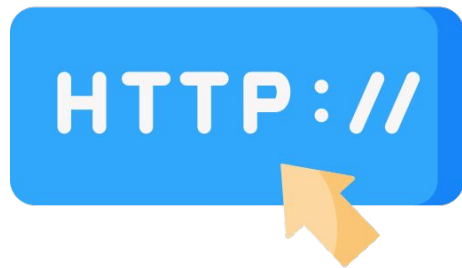




# El módulo HTTP

Este módulo es una parte integral del núcleo de Node.js y proporciona un conjunto de funcionalidades en todo lo que respecta, no solo a crear, sino también gestionar servidores web y aplicaciones cliente HTTP.

A continuación, veremos algunas de las propiedades y métodos más importantes del módulo:



# El módulo HTTP

Propiedades y Métodos	Descripción
<code>http.createServer()</code>	Este método permite crear un nuevo servidor HTTP. Toma una función de devolución de llamada como argumento que se ejecuta cada vez que el servidor recibe una solicitud.
<code>server.listen()</code>	Este método permite al servidor escuchar las solicitudes en el puerto especificado.
<code>server.on()</code>	Este método permite establecer un controlador de eventos para el servidor. Por ejemplo, se puede utilizar para manejar eventos como <i>"request"</i> (cuando se recibe una solicitud) y <i>"connection"</i> (cuando se establece una conexión).
<code>request.method</code>	Esta propiedad indica el método HTTP utilizado en la solicitud (por ejemplo, <i>"GET"</i> o <i>"POST"</i> ).
<code>request.url</code>	Esta propiedad indica la URL solicitada.
<code>response.writeHead()</code>	Este método permite escribir el encabezado de la respuesta HTTP.
<code>response.end()</code>	Este método permite finalizar y enviar la respuesta al cliente.

Estas propiedades y métodos, entre muchos más, son los que conforman el módulo HTTP. Trabajaremos inicialmente con algunos de ellos, para entenderlos bien.

# El módulo HTTP

**En la web de Node.js, encontraremos la documentación oficial que nos proveerá la información completa de todas las propiedades y métodos que disponemos en este módulo.**

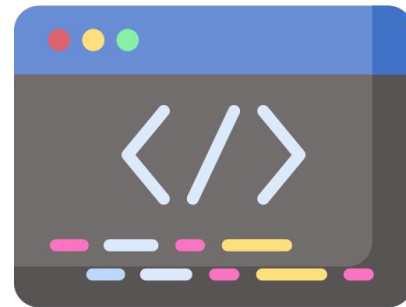
**No olvides consultarla ante cualquier duda que te surja.**

**<https://nodejs.org/api/http.html>**

# El módulo HTTP

Ya con el entorno listo para comenzar a programar y el principal actor que entrará en escena, comencemos a desarrollar el código base de nuestro primer servidor web.

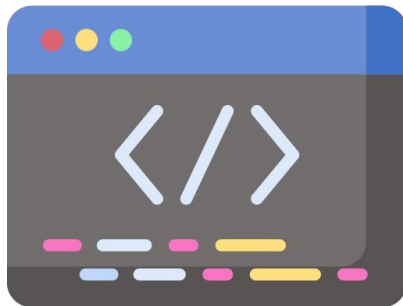
Trataremos este proceso como una receta, lo cual será la forma más fácil de organizar nuestras tareas.



# El módulo HTTP

## Nuestra receta consistirá en:

1. Sumar a nuestro proyecto Node, el servidor HTTP que necesitamos utilizar
2. Luego, definiremos un puerto donde el servidor “*escuchará*” peticiones
3. Crearemos el servidor web, e iniciaremos su escucha
4. Ante una petición, devolveremos una respuesta simple basada en texto plano



**UNTREF**

UNIVERSIDAD NACIONAL  
DE TRES DE FEBRERO

# El módulo HTTP

El método **require()** se ocupa de sumar el módulo HTTP a nuestro proyecto. Cumple una función similar a cuando instanciamos una clase u objeto.

El puerto que utilizaremos es el **3000**.

El método **createServer()**, crea el servidor.

Por último, el método **listen()** comienza a escuchar peticiones en el puerto que le definimos.



```
const http = require('http');
const PORT = 3000

const server = http.createServer((request, response) => {

})

server.listen(PORT, () => {
  console.log(`Servidor ejecutándose en el puerto: ${PORT}`);
})
```

# El módulo HTTP



```
const http = require('http');  
const PORT = 3000
```

El **puerto numérico** que definimos corresponde a un puerto específico de la computadora, habitualmente administrado por el sistema operativo, el cual se define para que una tarea o proceso escuche peticiones de otras aplicaciones (*llamadas clientes*).

# El módulo HTTP



```
const server = http.createServer((request, response) => {  
  
  })
```

El método **createServer()** recibe una función como parámetro y, a su vez, ésta recibe dos objetos como parámetros: **request** y **response**.

El primero de estos objetos (**request**), es el cual canaliza (*recibe*) las peticiones en cuestión de cada cliente (*navegadores web, apps, etc.*).

**response**, por su parte, es quien se ocupará de brindar las respuestas a el o los clientes que le realizan peticiones.



# El módulo HTTP



```
const server = http.createServer((request, response) => {  
  response.statusCode = 200;  
  response.setHeader('Content-Type', 'text/plain');  
  response.end('Hola, mundo!');  
})
```

**Nos queda por delante responder la petición** al cliente que la realizó. Junto a la respuesta **debemos informarle**, lo que se denomina **un código de estado**, qué **tipo de contenido** le estamos enviando y, finalmente, **la respuesta que entregamos** a su petición.

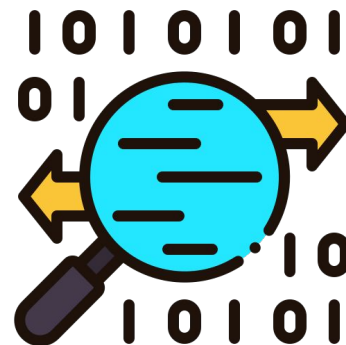
Todo esto, de la mano del objeto **response**.

# El módulo HTTP

Ya analizamos todo lo concerniente al código de nuestra primera aplicación web, de la mano de un servidor web.

Nos queda ponerlo en marcha y probar el mismo desde un navegador web, para así cerrar todo el circuito de trabajo

**Servidor Web/Cliente Web.**

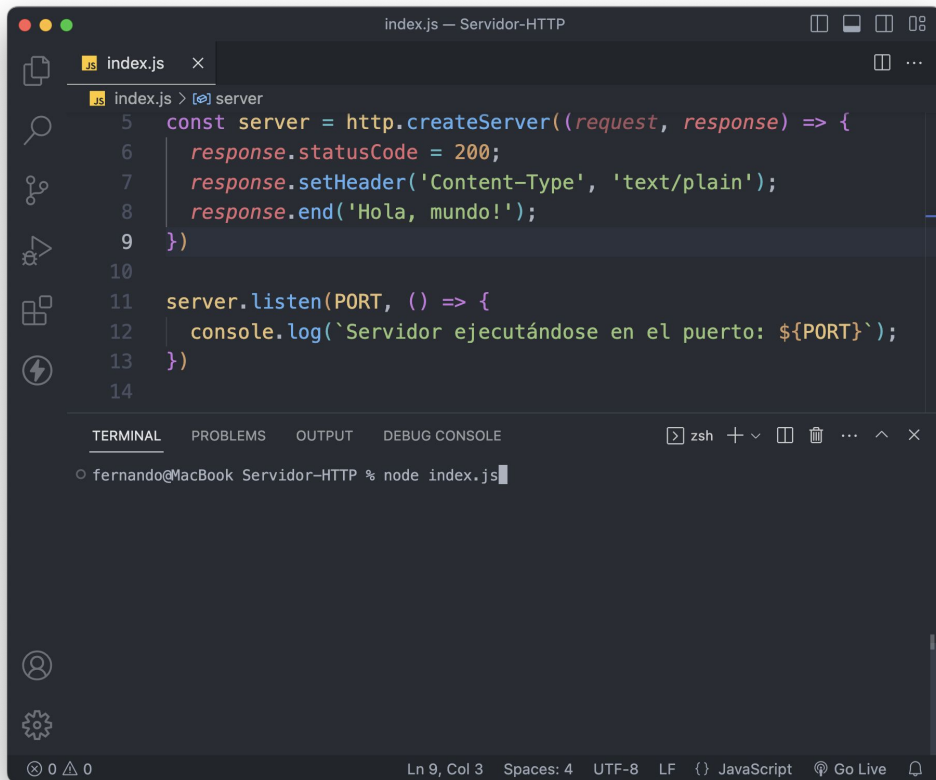


**UNTREF**

UNIVERSIDAD NACIONAL  
DE TRES DE FEBRERO

# Poner en funcionamiento nuestro servidor web

# Poner en funcionamiento nuestro servidor web



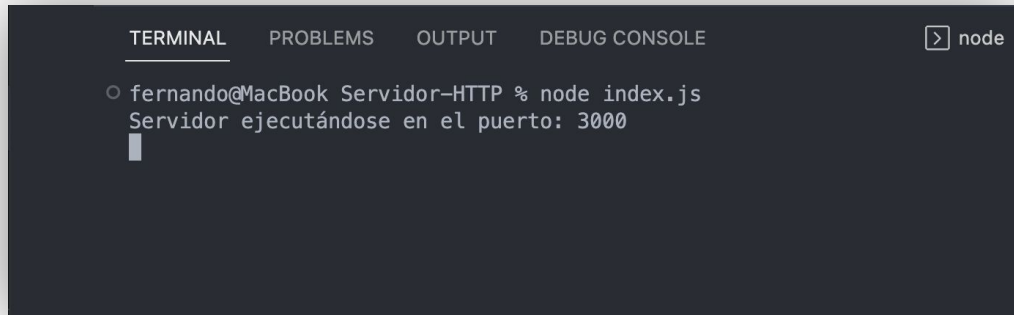
```
index.js — Servidor-HTTP
index.js > server
5  const server = http.createServer((request, response) => {
6    response.statusCode = 200;
7    response.setHeader('Content-Type', 'text/plain');
8    response.end('Hola, mundo!');
9  })
10
11  server.listen(PORT, () => {
12    console.log(`Servidor ejecutándose en el puerto: ${PORT}`);
13  })
14

TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE
fernando@MacBook Servidor-HTTP % node index.js
```

Para poner en marcha el servidor web, debemos utilizar la ventana Terminal y el comando **node**.

Como siempre, el parámetro que debe recibir es el nombre del archivo JavaScript donde está el código que le da vida al **webserver**.

# Poner en funcionamiento nuestro servidor web

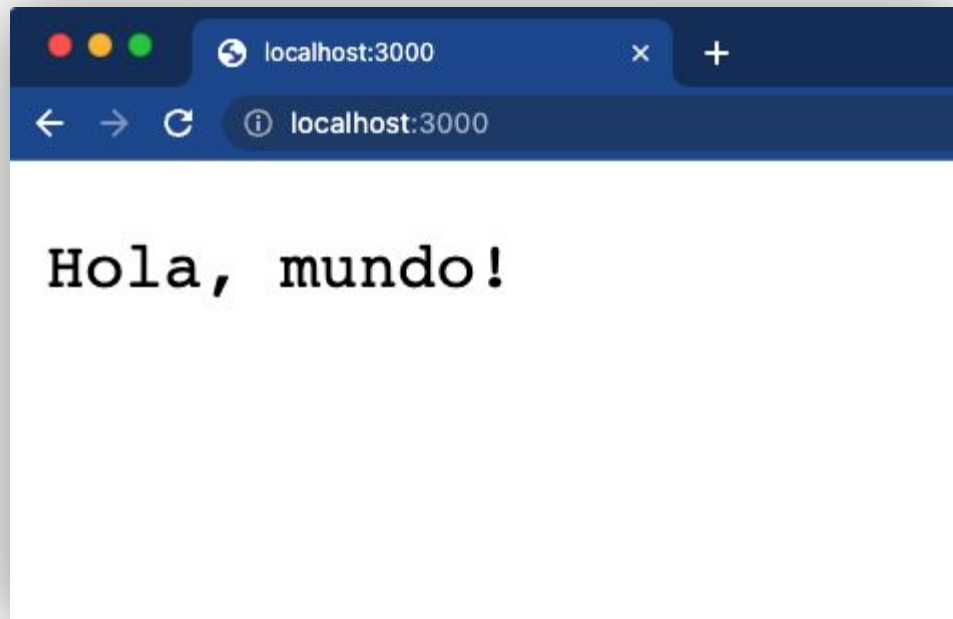


```
TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE  > node
fernando@MacBook Servidor-HTTP % node index.js
Servidor ejecutándose en el puerto: 3000
```

Al ejecutar este comando, en la ventana Terminal veremos el mensaje de log de nuestra aplicación, el cual nos informa en qué puerto se ejecuta el servidor.

**Nos vamos al navegador web para ejecutar la “Prueba de fuego 🔥”.**

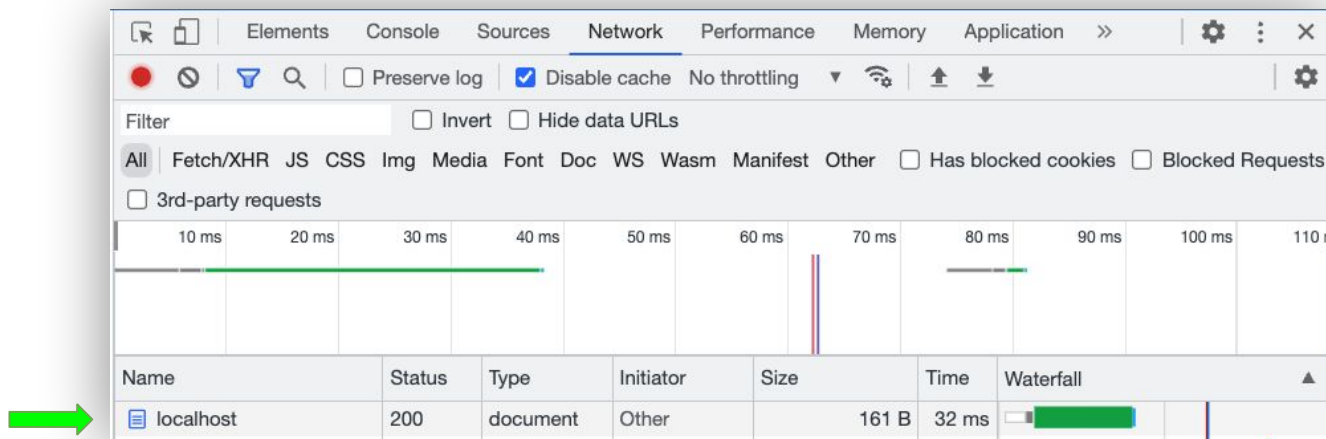
# Poner en funcionamiento nuestro servidor web



Escribimos en una nueva pestaña, la URL correspondiente a nuestro servidor web, y pulsamos Enter.

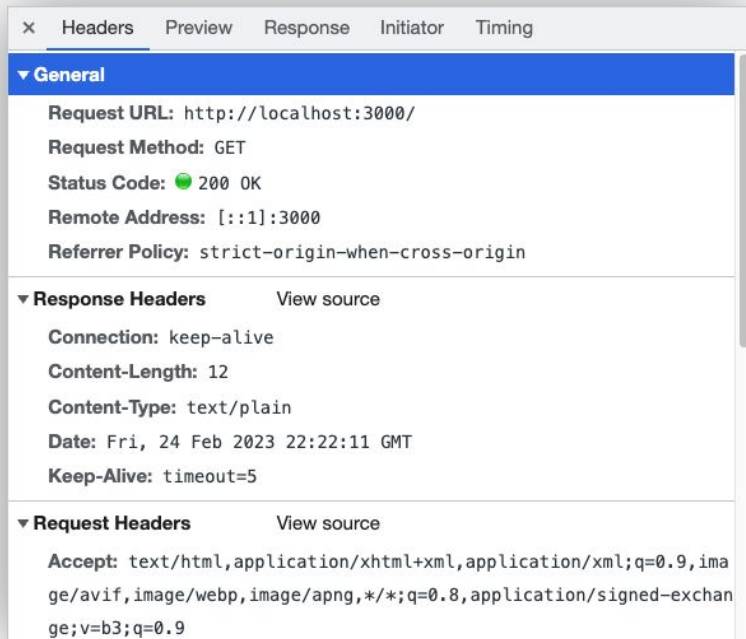
Si todo fue bien, debemos visualizar un texto simple, como el que agregamos en nuestra respuesta de servidor.

# Poner en funcionamiento nuestro servidor web



Abrimos **DevTools** en el navegador web y, en la pestaña **Network**, encontramos información relacionada a la respuesta recibida por parte del servidor: *Tiempo de respuesta*, *Status*, *Tipo de archivo de respuesta*, entre otros datos adicionales.

# Poner en funcionamiento nuestro servidor web



Seleccionando el archivo de respuesta del servidor, accedemos a un detalle mucho más amplio de *“todo el diálogo”* que ocurre entre el webserver y el cliente, en este caso, el navegador web en cuestión.

**UNTREF**

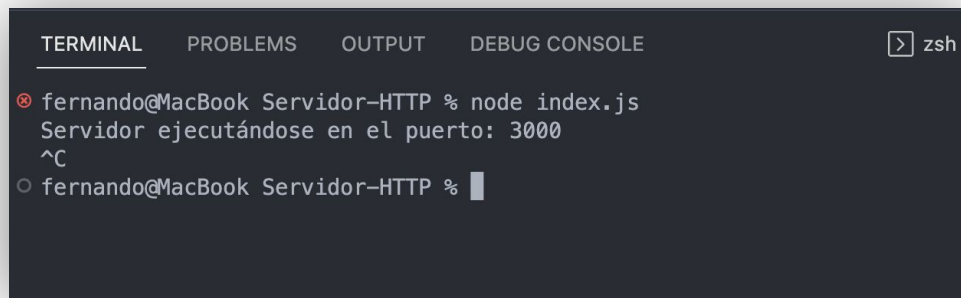
UNIVERSIDAD NACIONAL  
DE TRES DE FEBRERO



# Poner en funcionamiento nuestro servidor web

Para detener el webserver, debemos darle foco a la ventana Terminal y presionar la combinación de teclas **CTRL + C**.

Así interrumpimos la ejecución del mismo.

A screenshot of a terminal window with a dark background. The window has tabs at the top: 'TERMINAL' (selected), 'PROBLEMS', 'OUTPUT', and 'DEBUG CONSOLE'. In the top right corner, there is a 'zsh' icon. The terminal content shows a prompt 'fernando@MacBook Servidor-HTTP %' followed by the command 'node index.js'. The output is 'Servidor ejecutándose en el puerto: 3000'. Below that, the prompt is followed by '^C', indicating an interrupt signal. The final line shows the prompt 'fernando@MacBook Servidor-HTTP %' with a cursor.

# Herramientas adicionales

Existen otras tantas herramientas adicionales que nos permiten trabajar de forma más efectiva entre el “*ida y vuelta*” del webserver al cliente, y viceversa.

Las iremos incorporando más adelante, a medida que seguimos aprendiendo los diferentes jugadores que seguirán entrando en escena.

# Poner en funcionamiento nuestro servidor web

```
Server Web

const http = require('http');

const PORT = 3000

const server = http.createServer((request, response) => {
  response.statusCode = 200;
  response.setHeader('Content-Type', 'text/plain');
  response.end('Hola, mundo!');
})

server.listen(PORT, () => {
  console.log(`Servidor ejecutándose en el puerto: ${PORT}`);
})
```

El código completo del servidor web.

**UNTREF**

UNIVERSIDAD NACIONAL  
DE TRES DE FEBRERO

# Poner en funcionamiento nuestro servidor web

Los parámetros que recibe el método **createServer()** dentro de la función, son siempre un **request** (petición), y el otro parámetro la respuesta (**response**).



```
const server = http.createServer((request, response) => {  
  })
```

Estos objetos en cuestión, suelen ser abreviados como **req** y **res**. Podemos sumarnos a esta forma de definirlos, porque es la que vamos a encontrar habitualmente en cualquier ejemplo que busquemos por Internet.



```
const server = http.createServer((req, res) => {  
  })
```

# Espacio de trabajo

# Espacio de trabajo

Pongamos en práctica este repaso general de la sintaxis básica y moderna de JavaScript, a través de un conjunto de ejercicios.

**Tiempo estimado:** 20 minutos. 

```
const questions = [ 'dudas', 'consultas', '🤔' ]
```



```
> node gracias.js
```