



Introducción a Node

Node y Express (Parte II)

Describir las características fundamentales del entorno Node.js y su utilidad para el desarrollo de aplicaciones web.

- Unidad 1:
Introducción a Node
- Unidad 2:
Node y el gestor de paquetes
- Unidad 3:
Persistencia



Te encuentras aquí



¿Qué aprenderás en esta sesión?

- Reconocer el uso del enrutador de Express para la creación de rutas en el servidor.
- Construir un servidor con Express que disponibiliza una ruta por defecto para la devolución de una respuesta personalizada.

¿Qué características
especiales tiene Node?



¿Cuál es la ventaja de utilizar Express en comparación con Node puro para el desarrollo de servidores?



/* Enrutamiento (Routing) */

Enrutamiento (Routing)

El enrutamiento se refiere a determinar cómo una aplicación responde a una solicitud del cliente a un punto final particular, que bien puede ser una URI o una URL, además de poseer algún método de solicitud HTTP específico (GET, POST, PUT, DELETE, etc.).

La definición de ruta en Express toma la siguiente estructura:

```
app.METHOD(PATH, CALLBACK)
```

- “app” es una instancia de Express.
- “METHOD” es un método de solicitud HTTP en minúscula.
- “PATH” es el complemento de una ruta en el servidor.
- “CALLBACK”, también conocido como “handler”, es la función que se ejecuta cuando la ruta es consultada.

Demostración "Toc toc"



Ejercicio guiado

Toc toc

Desarrollar un servidor que disponibiliza una ruta GET /TocToc que al ser consultada devuelva el mensaje “¿Quién es?”, sigue los siguientes pasos para el desarrollo de este ejercicio:

- **Paso 1:** Crear una instancia de Express con una constante “Express”.
- **Paso 2:** Guardar en una constante “app” la ejecución de la instancia que creaste en el paso anterior. ¿Cómo es posible esto? Resulta que Express inicia como una función que al ser ejecutada devuelve un objeto con los diferentes métodos que ocuparemos a lo largo de este módulo.



Ejercicio guiado

Toc toc

- **Paso 3:** Ocupar el método “listen” de la constante “app” para definir en el primer parámetro el puerto 3000 y como segundo parámetro, una función callback que imprima por consola que el servidor fue inicializado en el puerto 3000.
- **Paso 4:** Ocupar el método “get” de la constante “app” para crear una ruta **/TocToc** que utilice como callback el parámetro “res” (response) y su método “send”.



Ejercicio guiado

Toc toc

Este método nos servirá para devolver un contenido de texto, no obstante también se podrá interpretar en los navegadores como HTML.

```
// Paso 1
const express = require("express");

// Paso 2
const app = express();

// Paso 3
app.listen(3000, () => {
  console.log("El servidor está inicializado en el puerto 3000");
});

// Paso 4
app.get("/TocToc", (req, res) => {
  res.send("¿Quién es?");
});
```



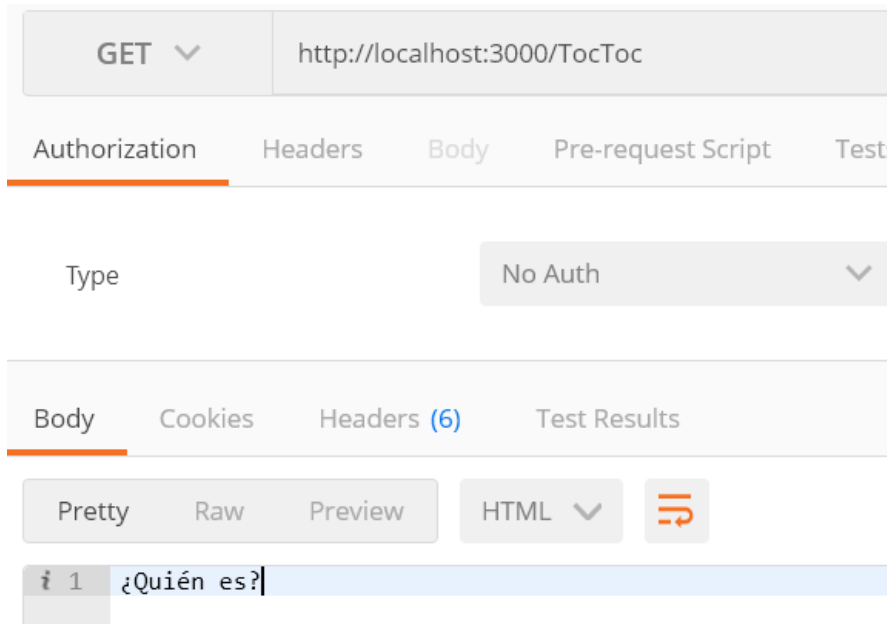
Ejercicio guiado

Toc toc

Levanta el servidor y
consulta la ruta

<http://localhost:3000/TocToc>

c utilizando POSTMAN y
deberás recibir lo que te
mostramos en la siguiente
imagen.

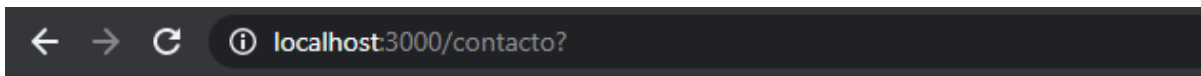


/* Ruta por defecto */

Ruta por defecto

Express evalúa cada una de las rutas (routes) que nosotros escribimos en el código, en el mismo orden en el que han sido escritas. ¿Pero qué pasa si un cliente envía una petición, por ejemplo <http://localhost:3000/contacto?>

Express devuelve el siguiente mensaje:



Cannot GET /contacto

Demostración
"Sorry, aquí no hay nada :/"



Ejercicio guiado

Sorry, aquí no hay nada :/

Incluir al servidor creado en la sección anterior una ruta genérica que devuelva el mensaje “Sorry, aquí no hay nada :/”, al recibir una petición a una ruta que no coincida con ninguna de las rutas creadas.

¿Cómo definimos una ruta genérica? Al igual que en otros lenguajes de programación, el carácter asterisco (*) nos sirve para definir un “Todo” o un “Cualquier cosa”, así que definiremos como “PATH” de esta ruta, un string con un asterisco. Su contenido se ejecutará entonces al consultar cualquier ruta indefinida por el servidor. En este caso la probaremos consultando la ruta:

<http://localhost:3000/MoteConHuesillo>.



Ejercicio guiado

Sorry, aquí no hay nada :/

Sigue los siguientes pasos para el desarrollo de este ejercicio guiado:

- **Paso 1:** Agregar una ruta GET y ocupa como primer parámetro lo siguiente: "*"
- **Paso 2:** Enviar como respuesta de la ruta genérica un encabezado centrado con HTML que diga "Sorry, aquí no hay nada :/"

```
const express = require("express");
const app = express();

app.listen(3000, () => {
  console.log("El servidor está inicializado en el puerto 3000");
});

app.get("/TocToc", (req, res) => {
  res.send("¿Quién es?");
});

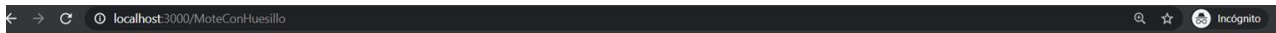
// Paso 1
app.get("*", (req, res) => {
  // Paso 2
  res.send("<center><h1>Sorry, aquí no hay nada :/ </h1></center>");
});
```

Ejercicio guiado

Sorry, aquí no hay nada :/

Ahora, abre tu navegador y consulta la ruta:

<http://localhost:3000/MoteConHuesillo>. Deberás recibir lo que te muestro en la siguiente imagen:



Sorry, aquí no hay nada :/

Consideraciones para las rutas genéricas

- Orden de definición: Las rutas genéricas deben ser ubicadas luego de la definición de todas las rutas, ya que al final, si Express no encuentra un handler adecuado para la petición recibida, usará este nuevo código de forma genérica para gestionar todas las peticiones que no coincidan con sus rutas preestablecidas.
- Patrón de ruta genérica: Al definir una ruta genérica, se puede utilizar un patrón de ruta más general, como una ruta raíz ("/") o un comodín ("*"), para capturar cualquier URL que no haya coincidido con las rutas predefinidas.

Consideraciones para las rutas genéricas

- Manejo de errores: Al utilizar rutas genéricas, es recomendable incluir un manejo de errores adicional para capturar y procesar las peticiones que no coinciden con ninguna ruta. Esto puede ayudar a devolver respuestas adecuadas o enviar mensajes de error personalizados en lugar de simplemente dejar que Express maneje la solicitud por defecto.
- Ordenamiento de rutas: Asegúrate de tener en cuenta el orden en que se definen las rutas genéricas y las rutas específicas. Si una ruta genérica está definida antes de una ruta específica que coincide con una URL dada, la ruta genérica será invocada en lugar de la ruta específica.

/* Objeto Request */

Objeto Request

Corresponde a la instancia, que contiene las propiedades definidas en una consulta emitida por una aplicación cliente. Con este objeto, podemos acceder a los parámetros, cuerpo o encabezados y es el primer parámetro de la función que se ejecuta, al coincidir con una ruta en el servidor, comúnmente denominado como “req”.

Este objeto contiene varios métodos y propiedades interesantes, pero los más utilizados son:

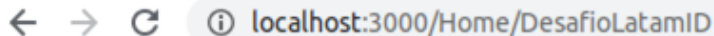
- **req.params:** Manejo de los parámetros recibidos por URL.
- **req.header():** Recibe como argumento el nombre de una propiedad y devuelve el valor de esta.
- **req.secure:** Confirma si el sitio de la consulta tiene el sello de seguridad https.
- **req.method:** Devuelve el método HTTP con el que se está haciendo la consulta.

Objeto Request

En el siguiente código te mostramos un ejemplo de cómo se vería una ruta GET /Home que recibe como subrecurso un parámetro "id":

```
app.get('/Home/:id', function (req, res) {  
  res.send(req.params.id)  
});
```

Como puedes ver, se está devolviendo el valor del mismo parámetro especificado en la ruta. Por ejemplo, si la ruta fuese <http://localhost:3000/Home/DesafioLatamID>, obtendremos lo siguiente:

A screenshot of a web browser's address bar. It shows navigation buttons (back, forward, refresh) and the address "localhost:3000/Home/DesafioLatamID".

localhost:3000/Home/DesafioLatamID

Demostración "Número de la suerte"



Ejercicio guiado

Número de la suerte

Crear un servidor con Express que escuche el puerto 3000. Una vez creado el servidor sigue los siguientes pasos para agregar una ruta GET /azar/:numero donde se obtenga el parámetro “numero” de la consulta y a la vez generar un número random entre el 1 y el 3.

El objetivo será responder al cliente un mensaje de éxito o fracaso en el caso que coincida o no con el número aleatorio:

- **Paso 1:** Agregar una ruta GET /azar/:numero.
- **Paso 2:** Ocupar el objeto Math para generar un número entero aleatorio entre 1 y 3.



Ejercicio guiado

Número de la suerte

- **Paso 3:** Utilizar la propiedad “params” del objeto request para guardar en una constante el parámetro “numero”.
- **Paso 4:** Utilizar un operador ternario para evaluar que el número generado de forma aleatoria, sea igual al recibido en la ruta como parámetro.



Ejercicio guiado

Número de la suerte

En caso de ser “true” la condición deberá devolver el mensaje “Hoy estás de suerte ;)”, en caso de ser false debes devolver el mensaje “Buena suerte para la próxima...”

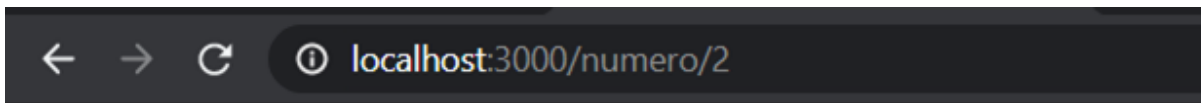
```
// Paso 1
app.get("/azar/:numero", (req, res) => {
  // Paso 2
  const n = Math.floor(Math.random() * (4 - 1)) + 1;
  // Paso 3
  const numero = req.params.numero;
  // Paso 3
  numero == n
    ? res.send("Hoy estás de suerte ;)")
    : res.send("Buena suerte para la próxima...");
});
```



Ejercicio guiado

Número de la suerte

Ahora intenta consultar desde el navegador varias veces la ruta que creaste definiendo un número como parámetro, y si tienes suerte, recibirás lo que te mostramos en la siguiente imagen:



Hoy estás de suerte ;)

/* Objeto Response */

Objeto Response

El objeto “response” representa la respuesta HTTP que enviamos desde el servidor cuando recibimos una solicitud HTTP. Por convención internacional, el objeto siempre se conoce como “res”.

¿Cuáles son las cualidades de este objeto? Entre sus métodos más comunes están:

- **res.download():** Permite la descarga de un archivo dentro del servidor. Es importante destacar que esta es una cualidad muy potente del framework Express.
- **res.redirect():** Así como su nombre lo indica, se utiliza para redireccionar la consulta recibiendo como argumento la dirección en formato string.
- **res.sendStatus():** Recibe como argumento el código de estado que deseemos devolver como respuesta de la solicitud.

Demostración

"¿Dónde puedo seguir estudiando?"

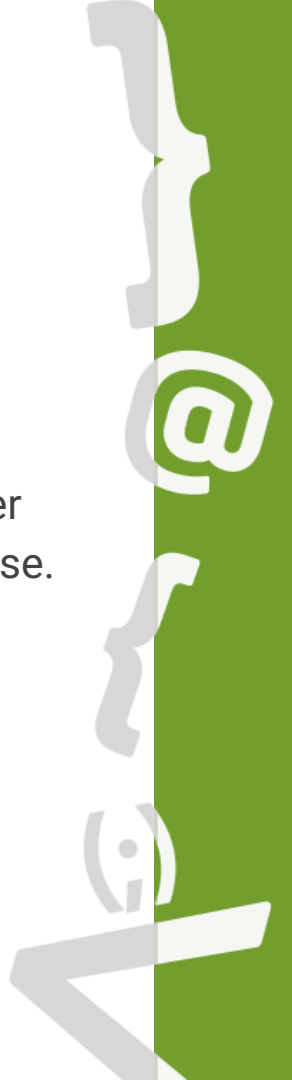


Ejercicio guiado

¿Dónde puedo seguir estudiando?

Crear un servidor con Express que escuche el puerto 3000. Una vez creado, el servidor sigue los siguientes pasos para agregar una ruta GET /estudiar que al ser consultada redireccione al usuario usando el método “redirect” del objeto response.

El objetivo será redirigirlo al sitio web de la academia [Desafío Latam.](#)



Ejercicio guiado

¿Dónde puedo seguir estudiando?

- **Paso 1:** Agregar una ruta GET /estudiar.
- **Paso 2:** Ocupar el parámetro res y el método “redirect” para redireccionar al cliente al sitio web de Desafío Latam.

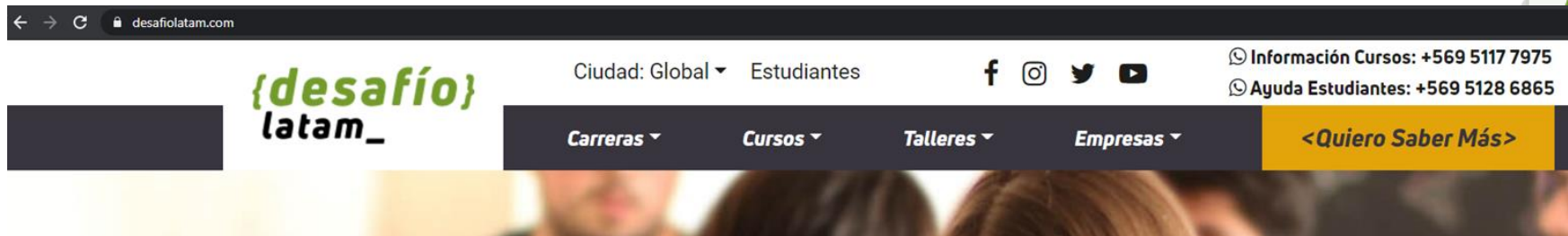
```
// Paso 1
app.get("/estudiar", function (req, res) {
  // Paso 2
  res.redirect("https://desafiolatam.com/");
});
```



Ejercicio guiado

¿Dónde puedo seguir estudiando?

Ahora levanta el servidor y consulta la ruta <http://localhost:3000/> estudiar y deberás ser direccionado a la página web de la academia, tal y como te mostramos en la siguiente imagen.



Resumen

- Express es una herramienta que permite crear rutas en una API REST utilizando la instancia de Express y los métodos proporcionados por ella.
- La función callback de cada ruta en Express tiene acceso a los parámetros "request" y "response", que permiten obtener las propiedades enviadas por el cliente y definir las propiedades para la respuesta de la solicitud.
- El enrutamiento en Express se refiere a cómo la aplicación responde a una solicitud del cliente a un punto final específico, utilizando métodos de solicitud HTTP específicos y una ruta definida.
- Es posible definir una ruta por defecto en Express para manejar las solicitudes a rutas no definidas, enviando un mensaje personalizado o un código de error genérico al usuario. Esta ruta genérica debe colocarse al final de todas las rutas definidas para que se ejecute cuando no se encuentra ninguna coincidencia con las rutas preestablecidas.

¿Cuáles son algunos
métodos comunes del objeto
response en Node.js y
Express?



¿Cómo se utiliza el método
"redirect" del objeto response
en Express para
redireccionar a una página
web?





Próxima sesión...

- *Reconocer el uso de los middlewares en el desarrollo de servidores con el framework Express.*
- *Construir un servidor que utilice middlewares para el procesamiento de una lógica antes de la ejecución de la ruta.*

{desafío}
latam_

*Academia de
talentos digitales*

