

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso

(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148

[www.sceu.frba.utn.edu.ar/e-learning](http://www.sceu.frba.utn.edu.ar/e-learning)

**Curso:**

# **Professional Webmaster**



**UTN.BA**  
UNIVERSIDAD TECNOLÓGICA NACIONAL  
FACULTAD REGIONAL BUENOS AIRES

**Centro de  
e-Learning**  
Secretaría de Cultura y Extensión Universitaria

Centro de e-Learning SCEU UTN - BA. Medrano 951 2do piso

(1179) // Tel. +54 11 7078- 8073 / Fax +54 11 4032 0148

[www.sceu.frba.utn.edu.ar/e-learning](http://www.sceu.frba.utn.edu.ar/e-learning)

Módulo 4:

# Aplicaciones web con Node.js

Unidad 2:

## Rutas y controladores



## Presentación

En esta unidad comenzamos a entender como Node.js trabaja el ruteo de peticiones y como envía las respuestas a los usuarios..



## Objetivos

Que los participantes logren...

- Conocer cómo recibe Express una petición..
- Entender y utilizar los métodos **POST** o **GET**
- Armar y entender la **clase express.Router** para hacer el ruteo de un sitio.



## Bloques temáticos

1. Rutas.
2. Controladores.

# 1. Rutas

En sitios web o aplicaciones web dinámicas, que accedan a bases de datos, el servidor espera recibir peticiones HTTP del navegador (o cliente).

Cuando se recibe una petición, la aplicación determina cuál es la acción adecuada correspondiente, de acuerdo a la estructura de la URL y a la información (opcional) indicada en la petición con los métodos **POST o GET**.

Dependiendo de la acción a realizar, puede que se necesite leer o escribir en la base de datos, o realizar otras acciones necesarias para atender la petición correctamente.

Express posee métodos para especificar qué función ha de ser llamada dependiendo del verbo HTTP usado en la petición (**GET, POST, DELETE, etc.**) y la estructura de la URL ("ruta"). También tiene los métodos para especificar qué plantilla ("**view**") o gestor de visualización utilizar, donde están guardadas las plantillas de HTML que han de usarse y cómo generar la visualización adecuada para cada caso.

La definición de ruta tiene la siguiente estructura:



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

Donde:

- **app** es una instancia de express.
- **METHOD** es un método de solicitud HTTP.
- **PATH** es una vía de acceso en el servidor.
- **HANDLER** es la función que se ejecuta cuando se correlaciona la ruta.

Los siguientes ejemplos ilustran las definiciones de rutas simples.

```
var express = require('express');  
  
var app = express();  
  
app.get('/', function(req, res) {  
  res.send('Hola Mundo!');  
});
```

Las primeras dos líneas incluyen (mediante la orden **requiere()**) el módulo de Express y crean una aplicación de Express. Este elemento se denomina comúnmente **app**, y posee métodos para el enrutamiento de las peticiones HTTP, configuración del 'middleware', y visualización de las vistas de HTML, uso del motores de 'templates', y gestión de las configuraciones de las aplicaciones que controlan la aplicación.

Las líneas que siguen en el código (las tres líneas que comienzan con `app.get`) muestran **una definición de ruta** que se llamará cuando se reciba una petición HTTP GET con una dirección ('/') relativa al directorio raíz.

La función 'callback' coge una petición y una respuesta como argumentos, y ejecuta un `send()` en la respuesta, para **enviar** la cadena de caracteres: "Hola Mundo!".

## Métodos de ruta

Un método de ruta se deriva de uno de los métodos HTTP y se adjunta a una instancia de la clase express.

El siguiente código es un ejemplo de las rutas que se definen para los métodos **GET** y **POST** a la raíz de la aplicación.

```
// GET method route
app.get('/', function (req, res) {
  res.send('GET > vamos a la página homepage');
});

// POST method route
app.post('/', function (req, res) {
  res.send('POST > vamos a la página homepage');
});
```

Express da soporte a los siguientes métodos de direccionamiento que se corresponden con los **métodos HTTP**: get, post, put, head, delete, options, trace, copy, lock, mkcol, move, purge, propfind, proppatch, unlock, report, mkactivity, checkout, merge, m-search, notify, subscribe, unsubscribe, patch, search y connect.

Hay un método de direccionamiento especial, `app.all()`, que no se deriva de ningún método HTTP. Este método se utiliza para cargar funciones de middleware en una vía de acceso para todos los métodos de solicitud.

En el siguiente ejemplo, el manejador se ejecutará para las solicitudes a `"/secret"`, tanto si utiliza **GET, POST, PUT, DELETE**, como cualquier otro método de solicitud HTTP soportado en el módulo http.





```
app.all('/secret', function (req, res, next) {  
  console.log('Accessing the secret section ...');  
  next(); // pasa el control al siguiente controlador  
});
```

## Vías de acceso de ruta

Las vías de acceso de ruta, en combinación con un método de solicitud, definen los puntos finales en los que pueden realizarse las solicitudes. Las vías de acceso de ruta pueden ser series, patrones de serie o expresiones regulares.

Estos son algunos ejemplos de vías de acceso de ruta basadas en series.



```
app.get('/', function (req, res) {  
  res.send('root');  
});
```

Esta vía de acceso de ruta coincidirá con las solicitudes a la ruta raíz, /.



```
app.get('/quienes', function (req, res) {  
  res.send('quienes');  
});
```

Esta vía de acceso de ruta coincidirá con las solicitudes a /quienes.



```
app.get('/random.text', function (req, res) {  
  res.send('random.text');  
});
```

Esta vía de acceso de ruta coincidirá con las solicitudes a /random.text.



```
app.get('/ab(cd)?e', function(req, res) {  
  res.send('ab(cd)?e');  
});
```

Esta vía de acceso de ruta coincidirá con /abe y /abcde.


## 2. Controladores

### Manejadores de rutas

Puede proporcionar varias funciones de devolución de llamada que se comportan como middleware para manejar una solicitud. La única excepción es que estas devoluciones de llamada pueden invocar `next('route')` para omitir el resto de las devoluciones de llamada de ruta. Puede utilizar este mecanismo para imponer condiciones previas en una ruta y, a continuación, pasar el control a las rutas posteriores si no hay motivo para continuar con la ruta actual.

Los manejadores de rutas pueden tener la forma de una función, una matriz de funciones o combinaciones de ambas,

Una función de devolución de llamada individual puede manejar una ruta. Por ejemplo:



```
app.get('/example/a', function (req, res) {  
  res.send('Hello from A!');  
});
```

Más de una función de devolución de llamada puede manejar una ruta (asegúrese de especificar el objeto `next`). Por ejemplo:



```
app.get('/example/b', function (req, res, next) {
  console.log('the response will be sent by the next function ...');
  next();
}, function (req, res) {
  res.send('Hello from B!');
});
```

## Métodos de respuesta

Método	Descripción
<b>res.download()</b>	Solicita un archivo para descargarlo.
<b>res.end()</b>	Finaliza el proceso de respuesta.
<b>res.json()</b>	Envía una respuesta JSON.
<b>res.jsonp()</b>	Envía una respuesta JSON con soporte JSONP.
<b>res.redirect()</b>	Redirecciona una solicitud.
<b>res.render()</b>	Representa una plantilla de vista.
<b>res.send()</b>	Envía una respuesta de varios tipos.
<b>res.sendFile()</b>	Envía un archivo como una secuencia de octetos.
<b>res.sendStatus()</b>	Establece el código de estado de la respuesta y envía su representación de serie como el cuerpo de respuesta.

## app.route()

Puede crear manejadores de rutas encadenables para una vía de acceso de ruta utilizando `app.route()`. Como la vía de acceso se especifica en una única ubicación, la creación de rutas modulares es muy útil, al igual que la reducción de redundancia y errores tipográficos. A continuación, se muestra un ejemplo de manejadores de rutas encadenados que se definen utilizando `app.route()`.

```
app.route('/book')
  .get(function(req, res) {
    res.send('Consigue un libro al azar');
  })
  .post(function(req, res) {
    res.send('Agregar un libro');
  })
  .put(function(req, res) {
    res.send('Actualiza un libro');
  });
```

## express.Router

Utilice la **clase `express.Router`** para crear manejadores de rutas montables y modulares. Una instancia Router es un sistema de middleware y direccionamiento completo; por este motivo, a menudo se conoce como una “miniaplicación”.

Creemos un archivo de direccionador denominado **`nosotros.js`** en el directorio de la aplicación, con el siguiente contenido:



```
var express = require('express');
var router = express.Router();

/* GET nosotros page. */
router.get('/', function(req, res, next) {
  res.send('Nosotros página');
});

module.exports = router;
```

A continuación, cargue el módulo de direccionador en la aplicación (**archivo app.js**)



```
var nosotrosRouter = require('./routes/nosotros');
...
app.use('/nosotros', nosotrosRouter);
```



## Bibliografía utilizada y sugerida

### Artículos de revista en formato electrónico:

**Express.** Disponible desde la URL: <https://expressjs.com/es/>

**Node.js** Disponible desde la URL: <https://nodejs.org/es/>