

Diplomatura en programación web full stack con React JS



Módulo 4:

Introducción Reactjs y Nodejs

Unidad 3:

Introducción Node



Presentación

En esta unidad nos introduciremos en el mundo de Node.js, el motor de Javascript que se ejecuta en los servidores. También comenzaremos a explorar Express, el framework más utilizado de Node.js para la creación de sitios y aplicaciones web.



Objetivos

Que los participantes logren...

- Comprender el ecosistema Node.js/JavaScript.
- Entender y utilizar los distintos paquetes de npm .
- Conocer y utilizar el framework Express.js .



Bloques temáticos

1. Introducción / Instalación.
2. Express.
3. Rutas.
4. Controladores.

1. Introducción/Instalación

Node.js es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Node.js usa un modelo de operaciones E/S (Entrada/Salida) sin bloqueo y orientado a eventos, que lo hace liviano y eficiente. El ecosistema de paquetes de Node.js, npm, es el ecosistema más grande de librerías de código abierto en el mundo.

Fue concebido como un entorno de ejecución de JavaScript orientado a eventos asíncronos y está diseñado para construir aplicaciones en red escalables.

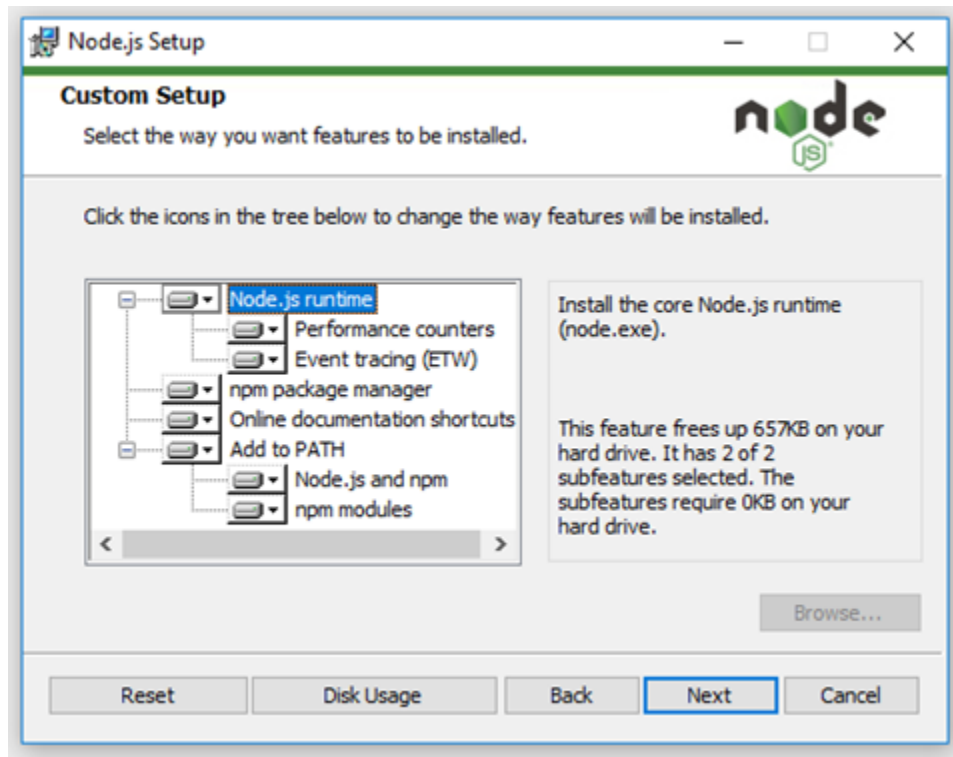
Node permite además la creación de sitios y aplicaciones web construidos enteramente usando JavaScript, así como también la creación y utilización de un sin fin de herramientas.

Instalación

Node.js (de aquí en adelante Node) se puede descargar desde su sitio web en <https://nodejs.org/es/>. La página principal nos ofrece 2 versiones, la LTS y la Actual. La principal diferencia entre estas es que la LTS (Long Term Service) va a recibir actualizaciones y parches de seguridad durante mucho más tiempo. En cambio la versión denominada "Actual" es la que incluye las últimas características del entorno.

Salvo que vayamos a instalar Node en un servidor para correr una aplicación web grande, en el uso diario se recomienda descargar e instalar la versión más nueva.

Durante la instalación es importante que tengamos habilitadas todas las opciones que se muestran a continuación



Una vez terminada la instalación podemos verificar que todo haya salido correctamente abriendo la consola (CMD) y escribiendo `node -v`. Este comando devuelve la versión instalada de Node, al momento de escribir esto es la **v14.16.1**

Si seguimos los pasos de la instalación deberíamos tener también instalado npm (Node Package Manager) que es el manejador de paquetes de Node, y es la herramienta que nos permitirá instalar todas las librerías de Javascript que necesitemos en nuestros proyectos.

Podemos verificar la instalación de npm escribiendo `npm -v` en la consola.

Uso básico de Node

En su forma más simple, podemos abrir la consola y escribir solamente `node`. Esto abrirá una instancia que nos permitirá ir ingresando expresiones de Javascript que serán evaluadas e impresas en el momento.

Algunos ejemplos de cosas que podemos probar:

- Operaciones matemáticas
- Asignaciones de variables
- Definición de funciones
- Y cualquier otro código Javascript válido

Para salir de la instancia de node escribimos **.exit** (con punto antes de la e) o presionamos la combinación de teclas CTRL+C.

También podemos ejecutar scripts directamente con Node. Creamos un archivo nuevo llamado index.js y escribimos el siguiente código:

```
var hoy = new Date( )

console.log('Hoy es ' + hoy)

var i
for(i = 0; i < 10; i++) {
    console.log(i)
}
```

Lo guardamos y lo ejecutamos escribiendo **node index.js**. El resultado debería ser la fecha y hora actual y un conteo de 0 a 9 como respuesta en la consola.

Manejo de paquetes de npm

Inicialización de un proyecto con Node.js

Para comenzar un proyecto con Node.js desde cero es necesario correr el script de inicialización `npm init`. Este comando debe ser corrido desde la consola, en la carpeta donde deseemos inicializar el proyecto.

El script nos hará una serie de preguntas básicas sobre nuestro proyecto. Podemos contestar todas apretando la tecla ENTER o llenar los datos que creamos necesarios.

Al finalizar, encontraremos un archivo llamado package.json donde se guardó toda la información que acabamos de cargar. Así mismo, en ese archivo, es donde el manejador de paquetes de Node, npm, va a ir guardando las dependencias o librerías de nuestro proyecto, así como también, varias de las herramientas que usaremos para desarrollarlo.

Instalación y uso de librerías con npm

La instalación de librerías se hace mediante el comando `npm install`. A este debemos indicarle qué librería deseamos que baje e instale y este lo hará de forma automática. Las librerías las podemos buscar en el sitio <https://www.npmjs.com/>, que es el repositorio central de npm.

A modo de ejemplo, vamos a instalar una librería llamada moment (<http://momentjs.com/>). Su página de npm es <https://www.npmjs.com/package/moment>, donde encontraremos información de la misma.

Ejecutamos el siguiente comando `npm install moment --save`. El argumento `--save` le indica a npm que deseamos que guarde la referencia a la librería como una dependencia en el archivo package.json. Una vez terminada la instalación podemos abrir este archivo para verificarlo.

2. Express

Es el framework web más popular de Node, y es la librería subyacente para un gran número de otros frameworks web populares de Node.

Proporciona mecanismos para:

- Escritura de manejadores de peticiones con **diferentes verbos HTTP** en diferentes caminos URL (rutas).
- Integración con motores de **renderización de "vistas"** para generar respuestas mediante la introducción de **datos en plantillas**.
- Añadir procesamiento de peticiones **"middleware"** adicional en cualquier punto dentro de la ejecución de una petición.

A pesar de que Express es en sí mismo bastante minimalista, los desarrolladores han creado librerías para trabajar con:

- Envío de mail: **nodemailer**
- Sesiones: **express-session**
- Base de datos: **mysql**


Instalación

Paso 1

Suponiendo que ya tenemos instalado **Node.js**, creamos un directorio para que contenga la aplicación y convertirlo en el directorio de trabajo.

Paso 2


Vamos a ejecutar una herramienta llamada `express-generator`. Esta se encarga de generar la plantilla básica de un proyecto de Express. Mediante el parámetro `--view` vamos a especificar que queremos usar Handlebars como motor de templates.



```
npx express-generator --view=hbs
```

Paso 3

También vamos a instalar **Nodemon**, que es el encargado de reiniciar automáticamente el servidor de desarrollo cada vez que cambiamos un archivo. Esto nos evita tener que estar constantemente reiniciándolo manualmente, lo que termina siendo tedioso y muy agotador.



```
npm i nodemon
```

Paso 4

Y por último con el comando `npm instalamos` todas las dependencias que `express-generator` dejó en el archivo `package.json`.



```
npm i
```



Paso 5

Modificamos el **archivo package**, para que el comando start ejecute nodemon en vez de node.

```
{  
  "name": "sitio-transportes-node",  
  "version": "0.0.0",  
  "private": true,  
  "scripts": {  
    "start": "nodemon ./bin/www"  
  }  
}
```

Paso 6

En la consola ejecutamos el comando **npm start**, luego vamos a un navegador y escribimos: **localhost:3000** para ver sitio.

3. 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.

4. 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
res.download()	Solicita un archivo para descargarlo.
res.end()	Finaliza el proceso de respuesta.
res.json()	Envía una respuesta JSON.
res.jsonp()	Envía una respuesta JSON con soporte JSONP.
res.redirect()	Redirecciona una solicitud.
res.render()	Representa una plantilla de vista.
res.send()	Envía una respuesta de varios tipos.
res.sendFile()	Envía un archivo como una secuencia de octetos.
res.sendStatus()	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/>

Handlebars. Disponible desde la URL: <https://handlebarsjs.com/>