



# Construcción de API RESTful Con Node.js

## Breve descripción:

En este componente formativo se hará uso de Node.js para crear una API RESTful que se comunique con una base de datos MongoDB para permitir la ejecución de algunas operaciones CRUD.

---

Agosto 2023

## Tabla de contenido

Introducción .....	1
1. Definiciones .....	2
2. Generalidades de Node.js .....	5
2.1. Instalación de herramientas.....	6
2.2. Dependencias .....	17
2.3. Comandos y direccionamiento básico.....	19
3. Construcción API.....	21
3.1. Conexión Node.js y MongoDB .....	21
3.2. Construcción de consultas con Node .....	31
Síntesis .....	50
Material complementario.....	51
Glosario .....	52
Referencias bibliográficas .....	53

## Introducción

Le damos la bienvenida a este componente formativo en el que se estudiará la construcción de API RESTful con Node.js; para conocer la importancia del tema y los diferentes componentes que se incluirán, le invitamos a revisar el siguiente video:

### Video 1. Construcción de API RESTful con Node.js



[Enlace de reproducción del video](#)

#### Síntesis del video: Construcción de API RESTful con Node.js

Hola estimado aprendiz, el día de hoy te quiero acompañar en con componente formativo Construcción de API RESTful Con Node.js.

Si bien las aplicaciones de hoy en día necesitan el acceso a datos, es una muy buena práctica conectar nuestras bases de datos a las aplicaciones, ya que se crean

unas dependencias, unas interdependencias muy fuertes y en el momento en que queramos evolucionar estas aplicaciones o avanzar en la tecnología pues esto nos va a restringir muchas posibilidades; la opción correcta es construir Apis, en este caso APIS RESTful que van a ser intermediarios que van a permitir que toda la lógica asociada a la conexión a la base de datos y a la administración de la misma quede aislada de la aplicación para que solo se enfoque en su lógica.

Es una práctica muy recomendada que profesionalmente se utiliza todo el tiempo.

En este componente formativo abarcaremos las generalidades de lo que son las API REST y hablaremos de Node.js que es nuestra plataforma para construir servicios de “backend” y finalmente abordaremos componentes específicos para construir nuestra primera API REST utilizando esa tecnología.

Espero que el contenido sea de tu agrado.

## **1. Definiciones**

Como hemos mencionado, la mayoría de los sistemas de información requieren la gestión y transformación de datos, que, generalmente, se almacenan y protegen en sistemas gestores de bases de datos; sin embargo, se pueden presentar diferentes situaciones en las que se necesite refactorizar las funcionalidades ya establecidas, por lo que el uso de API RESTful es la respuesta para efectuar tales actualizaciones sin afectar el desempeño normal del sistema.

Entonces,

### **¿Qué es una API?**

“Application Programming Interfaces” (API), que traducido literalmente al español significa Interfaz de programación de aplicaciones; es esencialmente un conjunto de especificaciones, reglas y/o mecanismos que se definen para establecer la comunicación entre dos componentes de “software”.

Así, una API de REST o conocida también como API RESTful es una interfaz que permite la comunicación entre dos componentes de “software”, usualmente, entre un sistema gestor de base de datos y una aplicación de “software”, que la consume.

Hasta aquí hemos agregado otro término, REST:

### **¿A qué hace referencia?**



Una de las características de REST consiste en el no almacenamiento del estado; es decir, cada vez que se realiza una solicitud de transmisión, el emisor se debe identificar por medio de credenciales o “tokens”, ya que esta información no se almacena de ninguna forma. Aunque esto parece una desventaja, se encuentra que, al no tratar con el manejo de memoria, se disminuyen los procesos requeridos para el procesamiento de los datos y lo hace altamente compatible para ser utilizado por cualquier tecnología y para entornos con necesidades de escalamiento horizontal.

Por medio de una API REST se pueden implementar los HTTP Verbs o los métodos HTTP, por medio de los cuales se especifica el tipo de solicitud que se realiza al servidor y así este puede dar el tratamiento adecuado y ofrecer respuestas correctas.

Veamos cada uno de los métodos HTTP y su función o uso:

- I. **GET:** consulta de información al servidor, equivale a una solicitud find() en NOSQL o SELECT en SQL.
- II. **POST:** este método se usa para el envío de información al servidor generalmente para realizar un proceso de registro de datos, obligatoriamente implica el envío de datos adicionales en la solicitud. Es equivalente a un comando tipo insertOne() o insertMany() en NoSQL o la sentencia INSERT en SQL.
- III. **PUT:** este método es similar al anterior, pero se utiliza para la actualización de datos del lado del servidor. Es equivalente a un comando updateOne() o updateMany() en NOSQL o una sentencia UPDATE en SQL.
- IV. **PATCH:** este método también se utiliza para la actualización de registros en la base de datos, pero solo cuando la actualización es parcial y no del registro completo.
- V. **DELETE:** método utilizado para la eliminación de registro en el servidor. Es equivalente al comando deleteOne() o deleteMany() en NoSQL y a la sentencia DELETE de SQL.
- VI. **HEAD:** método que sirve para obtener información general de los recursos del servidor como, por ejemplo, validar la accesibilidad. No sirve para la gestión directa de registros propios del servidor.

Ahora bien, según Red Hat (2020) una API RESTful debe cumplir con los siguientes criterios:

**Criterio 1.** Debe usar arquitectura tipo cliente-servidor con manejo de solicitudes por medio del protocolo HTTP.

**Criterio 2.** Manejo de comunicaciones entre cliente y el servidor sin almacenar el estado.

**Criterio 3.** Los datos pueden almacenarse en caché para operaciones clave y así eliminar interacciones entre el cliente y el servidor.

**Criterio 4.** Arquitectura en capas que separe las solicitudes de los clientes y las respuestas del servidor.

**Criterio 5.** Debe existir una interfaz uniforme de comunicación.

## 2. Generalidades de Node.js

JavaScript es, tal vez, uno de los lenguajes más comunes en el desarrollo de aplicaciones web; generalmente, este lenguaje permite enriquecer el contenido que se muestra a los clientes por medio de un navegador web y junto a HTML y CSS completan el paquete completo de elementos que se debe combinar para ofrecer una buena experiencia de navegación por sitios WEB.

Así, es importante que conozcamos a qué hace referencia los términos HTML, CSS y JavaScript.

Como observamos, JavaScript se ejecuta del lado del cliente, esto significa que utiliza los recursos del equipo de cómputo del usuario que usa el navegador web, para controlar muchos de los procesos de validaciones y de gestión de datos. En este sentido, se podría decir que Node.js es la versión de JavaScript que puede gestionar la ejecución de código del lado del servidor como lo hacen los lenguajes PHP, ASP.NET, JSP, entre otros.

Conozcamos algunas de las características de Node.js y JavaScript:

- A. Tiene código abierto.** Node.js en un entorno de ejecución “Open Source” (código abierto), multiplataforma.
- B. Orientado a eventos asincrónicos.** Se orienta a eventos asíncronos que puede ejecutar código JavaScript por fuera de los entornos de un navegador web.
- C. Es tendencia.** Actualmente, Node.js se posiciona como una tendencia muy importante en el mundo laboral relacionado con estas tecnologías web.

Según la encuesta Stack Overflow (2021), desde hace nueve años, JavaScript es el lenguaje de programación más utilizado por desarrolladores profesionales; asimismo, Node.js se ubica en la sexta posición de uso en esta misma encuesta, con lo cual se marca una tendencia.

## 2.1. Instalación de herramientas

Para la instalación de Node.js se recomienda acceder a su página oficial de descargas ubicada en la dirección: <https://nodejs.org/es/download>, en donde se



presentan distribuciones para los sistemas operativos Windows y MacOS e incluso con el código fuente, recordemos que esta es una plataforma “open source”.

En la siguiente figura se observan las diferentes posibilidades de instalación habilitadas actualmente:

**Figura 1.** Visiones instaladoras Node.js



The screenshot shows the Node.js download page. At the top, there are two tabs: 'LTS' (Recommended for most) and 'Actual' (Latest features). Below these, there are three main installation options: 'Instalador Windows' (node-v16.15.0-x86.msi), 'Instalador macOS' (node-v16.15.0.pkg), and 'Código Fuente' (node-v16.15.0.tar.gz). Below these, there are two tables. The first table lists various installation options for Windows, macOS, and Linux. The second table lists additional platforms like Docker, Power LE Systems, System z, and AIX.

	32-bit	64-bit
Instalador Windows (.msi)	32-bit	64-bit
Binario Windows (.zip)	32-bit	64-bit
Instalador macOS (.pkg)	64-bit / ARM64	
Binario macOS (.tar.gz)	64-bit	ARM64
Binario Linux (x64)	64-bit	
Binario Linux (ARM)	ARMv7	ARMv8
Código Fuente	node-v16.15.0.tar.gz	

	Imagen Docker Oficial Node.js
Imagen Docker	Imagen Docker Oficial Node.js
Linux en Power LE Systems	64-bit
Linux en System z	64-bit
AIX en Power Systems	64-bit

Nota. <https://nodejs.org/es/download/>

Como se observa en dicha figura, la página oficial recomienda la descarga de versiones LTS (“Long Term Support”), lo que significa que estas versiones tienen

soporte de largo plazo y un funcionamiento estable, altamente validado y con gran cantidad de documentación asociada.

Es importante anotar que este campo del desarrollo “software” es muy dinámico, con permanentes actualizaciones que generan diferentes versiones; así, la última versión LTS de Node.js es la 16.15.0 y la de Node.js corresponde a la 18.1.0. Sin embargo, aunque estas herramientas son tan activas, presentan algunas generalidades que son esenciales conocer, con las cuales se garantiza su funcionamiento independiente de la versión que se descargue.

Adicionalmente, se requiere de un IDE de programación que permita el cargue y la ejecución de los paquetes de Node.js. En este punto, puede hacer uso de cualquier IDE que utilice normalmente para el desarrollo de aplicaciones afines a la web, particularmente se recomienda el uso de “Visual Studio Code”.

Este es un editor de código fuente que puede ser utilizado para el desarrollo de lenguajes como Python, Java, PHP, HTML/CSS, TypeScript, YAML y, por supuesto, JavaScript; tiene la posibilidad de ampliar sus funciones con “plugins” y buscadores de otros “plugins”, lo que permite una mayor personalización, según las necesidades de los usuarios, sin afectar su condición de código muy potente y liviano.

Las siguientes son algunas de las características importantes de este editor de código:

- a.** Soporte para depuración.
- b.** Integración con sistemas de versionamiento como GIT.
- c.** Verificación de sintaxis.
- d.** Ayudas de finalización de código inteligente.

e. Opciones de refactorización de código.

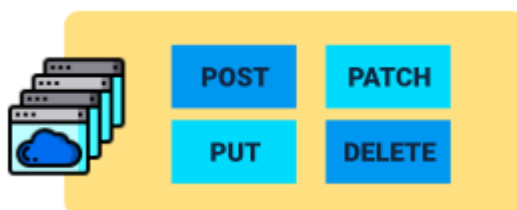
“Visual Studio Code” puede ser descargado de forma gratuita directamente desde su página oficial: <https://code.visualstudio.com/Download>, está disponible para sistemas operativos Windows, versiones de Linux Debian, Ubuntu, Red Hat, Fedora y SUSE y sistema operativo MacOS.

**Figura 2.** Versiones instaladoras “Visual Studio Code”



Nota. <https://code.visualstudio.com/Download>

Por último, se requiere de un sistema que nos permita hacer uso de la API REST para verificar su funcionamiento; recordemos que la API RESTful es un intermediario entre un sistema de base de datos y una aplicación que hace uso de estos servicios. Si bien las solicitudes que se realizan a la API se hacen por medio de HTTP, solo las solicitudes de tipo GET pueden ser verificadas por medio de un navegador web.



Recordemos que hay otros tipos de métodos que requieren de un mecanismo que permita identificar los parámetros y el tipo de método para ser procesado en el servidor.

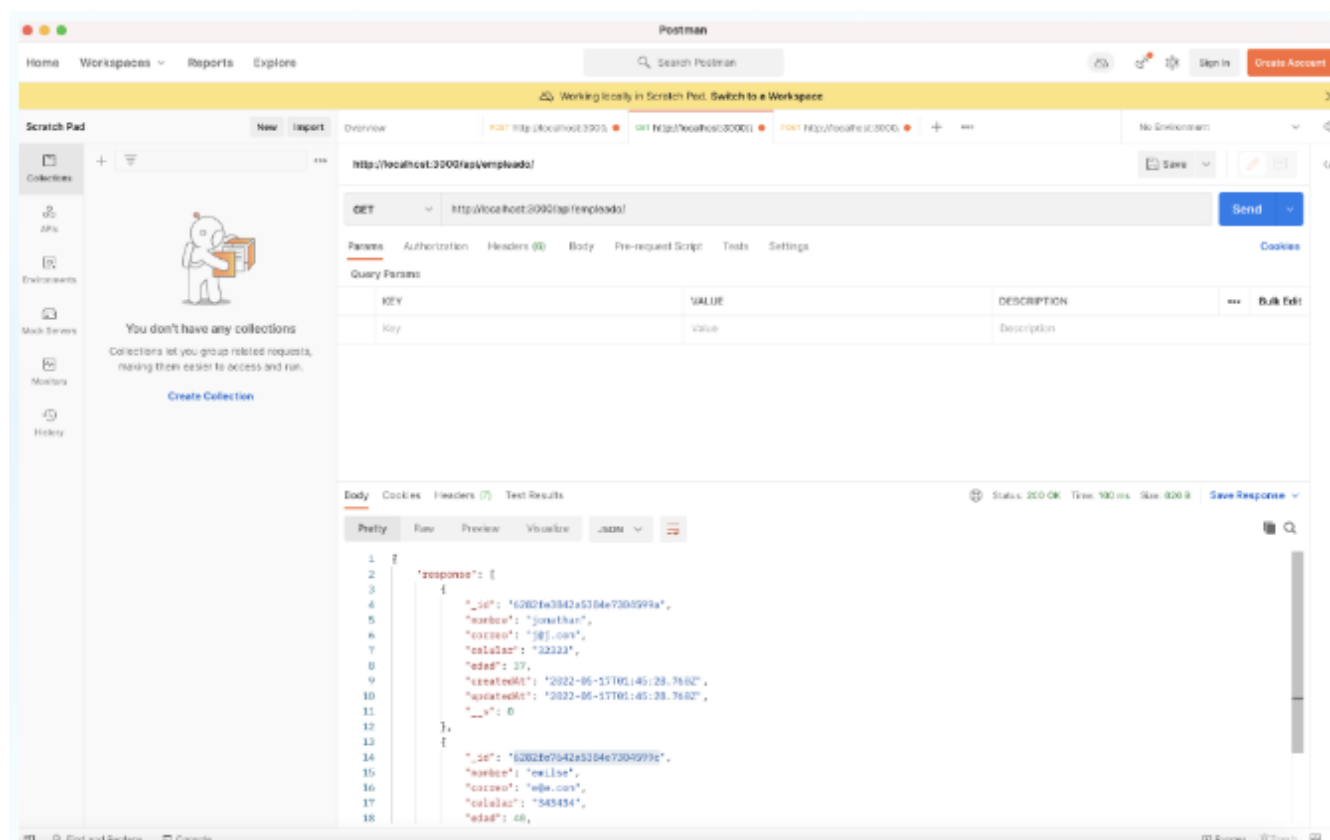
Para suplir esta necesidad, se recomienda instalar Postman, plataforma que facilita la construcción y prueba de APIs con una versión libre, con las funcionalidades necesarias para la verificación del funcionamiento de todos los métodos HTTP de nuestras API RESTful.

Puede utilizar esta plataforma de dos formas:

**01.** Descargue la versión desktop desde la página oficial disponible en:

<https://www.postman.com/downloads/>

**Figura 3.** Interfaz principal de Postman



Nota. <https://www.postman.com>

**02.** Realice el registro en línea, ingresando a la página oficial disponible en:

<https://www.postman.com>, en este caso, le invitamos a revisar el siguiente video, en el que se presenta el proceso para la instalación y configuración de Postman:

## Video 2. Instalación de Postman



### [Enlace de reproducción del video](#)

#### **Síntesis del video: Instalación de Postman**

Hola, ahorita vamos a dedicarnos a hablar un poco de Postman, entonces nosotros para poder probar nuestros servicios de backend necesitamos una herramienta que nos permita hacer la verificación de que los servicios que estamos publicando a través de esta API REST realmente están funcionando.

Recordemos que las APIS REST van a hacer consumidas por una aplicación externa, pero no tenemos que esperar hasta que la aplicación esté completa para poder comprobar sus servicios; postman es una herramienta que me permite a mí

hacer este proceso, entonces yo puedo entrar a la página de postman oficial, [www.postman.com](http://www.postman.com) y voy a poder encontrar este tipo de interfaz.

Postman como tal tiene dos formas de trabajo una forma de trabajo “online” y otra forma de trabajo local, para trabajar “online” o local digamos que siempre es importante registrarnos por una cuenta de correo valida, en la esquina superior derecha encontramos “Sign On four free” y aquí vamos a poder hacer el registro, podemos crear un registro utilizando un “e-mail”, un “password” y una contraseña, o podemos sencillamente iniciar sesión utilizando Google. Voy a utilizar esta opción, voy a utilizar una cuenta de correo que tengo de prueba, hay una licencia que debo leer, que debo aceptar, luego aquí nos pregunta un poco acerca de nosotros, esto es totalmente digamos para personalizar un poco lo que viene siendo la interfaz de postman, vamos a colocar acá que somos “backend developer”, bien, que, si queremos invitar más personas, vamos a seguir sin ningún equipo vamos a trabajar de forma individual y aquí me aparece la interfaz inicial de postman.

Esta es la interfaz web, es muy parecida a la de escritorio, básicamente aquí encontramos un conjunto de opciones que se pueden explorar en detalle, sencillamente les voy a mostrar la forma fácil de encontrar los servicios; lo primero es que yo para poder entrar a consumir una API, pues necesito un espacio entonces voy a My Workspace, entonces en el Workspace aquí doy clic en el signo más voy a empezar a hacer solicitudes. Esto que esta acá es lo que vamos a ver también exactamente igual en la versión de postman de escritorio.

Bueno, pero hemos hablado de API pero como hacemos para poder consumir una API y poder probarla a través de postman, entonces voy a abrir aquí, en internet

busque una API REST de ejemplo, por ejemplo esta que esta acá que es de gores.gov.in, como estas hay muchas publicadas de forma gratuita simplemente para hacer pruebas, como no hemos construido nuestros servicios, queremos verificar que postman si está configurado y está funcionando bien.

¿Entonces acá me dice que hay un servicio que esta publicado en esta dirección o URL cierto? Que es para obtener usuarios, entonces acá hay unas solicitudes que son get, unas solicitudes de son post, put, pach, miren que aquí están todos los métodos que se han comentado, métodos HTML, métodos que están cubiertos por esta API, esta API REST; entonces en el caso de los servicios que son GET, yo puedo utilizarlos directamente del navegador para poder consumirlos, de hecho si entro a la dirección el navegador me sirve para mostrar esta información, que esta información viene en un formato json, formato json en este caso. Pero también puedo hacer lo mismo, puedo utilizar la URL de este servicio que esta publicado y verlo a través de postman, entonces aquí en esta parte puedo ver los diferentes tipos de métodos http que puedo utilizar, en este caso como es un método GET simplemente necesito colocar la URL, pegar la URL del servicio que quiero probar y hago la solicitud del envío.

Envía la solicitud y aquí tengo en la parte inferior le respons o la respuesta al servicio, me lo da un poquito más bonito, más fácil de entender, pero digamos que la potencia realmente de postman no va solamente con las solicitudes get, eventualmente yo voy a tener que hacer solicitudes de tipo post, de tipo delete, de los diferentes tipos que necesitan un poco más de argumentos y son un poco más estructurados y no los puedo probar directamente del navegador. Entonces volviendo acá miramos que por ejemplo miramos que este empoint de acá es un método post



que me permite a mi crear un usuario, entonces voy a colocar esto en postman, voy a indicarle que es una solicitud post, voy a mandarle la URL en este caso pero como es un post normalmente se requieren de parámetros, entonces miremos acá este me sirve para crear un nuevo usuario si, y adicional a esto acá hay una anotación muy importante, en todos los casos no pasa, pero en este caso por utilizar los métodos post, patch, delete, necesitan un token de acceso basado como autorización, esto es una práctica muy común cuando se quiere proteger de alguna forma se colocan algún tipo de elemento de autorización, algunos gestores de acceso con algunos “tokens” para poder acceder a él. Su utilizamos este mismo ejemplo pues acá hay una opción para tener ese “token” yo ya la tengo acá lista, ya lo solicite entonces me dieron este “token”, entonces lo que voy a hacer es solicitar información de un servicio post para registrar un nuevo usuario, entonces saco url “token”, vuelvo a postman, miren que acá hay un conjunto de elementos que dicen parámetros, autorización, headers, aquí yo puedo configurar todo como se va a enviar la solicitud; entonces en este caso yo necesito una autorización, en la documentación de este servicio particular me está especificando eso, me está especificando que debo especificar que el headers es de tipo “Bearer token” y lo debo pasar como parámetro.

Entonces volviendo acá voy aquí a autorización, específico el tipo es un bearer “token” y aquí coloco el “token” listo.

Ahora que pasa, los parámetros también los requiero, como yo voy a enviar a registrar un nuevo usuario para este caso es una información que va a enviarse a una base de datos que seguramente puede ser NoSQL, sería un insert one por ejemplo yo veo que aquí los usuarios tienen un título, tienen unas características acá, vamos a enviar la solicitud tal como está para que miremos cuando se genera errores, mire

que yo acá envié la solicitud pero no le estoy pasando parámetros entonces la respuesta que me envía el servidor es: oiga no se pueden enviar estos campos vacíos se supone que esos son parámetros que tú me debes enviar en la solicitud que yo debo recibir.

Entonces acá en parámetros voy a colocar las llaves, recordemos que aquí hablamos de clave - valor, entonces el campo se llama field, el fiel es el e-mail, voy a colocar que voy a enviar un e-mail, me falta enviar un nombre, me falta enviar un género y me falta enviar un status; entonces aquí especifico el valor, voy a colocar aquí los valores y digo send. En este momento miren que la solicitud me indica a mi que he tenido una respuesta satisfactoria he podido llegar a hacer una inserción a través de este servicio con un método post especificándole los parámetros.

Entonces en este momento vamos a construir nuestros propios servicios de backend, vamos a generar nuestros propios endpoints y esta herramienta postman es la que me va a verificar si esos servicios que ya he publicado realmente están respondiendo a los mensajes que estoy enviando desde una solicitud en este caso una solicitud http.

Ahora si yo necesito instalar postman versión local que sería básicamente el mismo funcionamiento pues yo puedo irme también aquí a home y aquí hay una opción para descargar en la parte inferior izquierda descargar la versión desktop o sencillamente entro a la dirección [postman.com/download](https://postman.com/download) y aquí me aparecen las opciones de descarga, en este caso pues está detectando automáticamente que estoy en un equipo Mac y me da la opción para descargar.

Una vez se descarga sencillamente vuelve a la interfaz en la que le solicita información tal cual como paso en la web, en la versión online, me pide información de cuenta de correo sencillamente que tipo de rol tengo y automáticamente paso a utilizar la herramienta, entonces es muy sencillo esto es una herramienta imprescindible entonces los invito a que hagamos el proceso de instalación de postman en nuestras máquinas para que podamos continuar.

## 2.2. Dependencias

Una de las grandes potencialidades de Node.js es precisamente la facilidad de acoplar y reutilizar paquetes de JavaScript públicos que agilizan y disminuyen el tiempo requerido para la creación de servicios de alta calidad en muy poco tiempo. Node.js utiliza NPM para lograr estos resultados.

### ¿Qué significa y cómo opera NPM?

**“Node Package Manager” (NPM)** es el administrador de paquetes de Node y es el encargado de realizar las tres actividades fundamentales (npm, Inc):

- I. Ayuda a los desarrolladores JavaScript a compartir fácilmente módulos de código empaquetados.
- II. Sirve de colección pública de paquetes de código fuente abierto para Node.js.
- III. Sirve como interfaz de línea de comandos que permite instalar y publicar esos paquetes.

A continuación, se detallan algunos de estos paquetes que serán usados en la construcción de la API RestFul:

- a. **Express:** se considera un “framework” de Node.js minimalista y flexible que facilita las operaciones requeridas para la construcción de una aplicación como son el enrutamiento de URL, gestión de sesiones, cookies.
- b. **Mongoose:** librería Node.js basada en modelado de datos orientados a objetos para MongoDB muy similar a un ORM en sistemas de bases de datos relacionales. Mongoose permite la aplicación de un esquema específico desde la capa de aplicación y facilita la ejecución de operaciones en bases de datos NOSQL como establecer la conexión y la ejecución de comando.
- c. **Body - Parser:** es común intentar hacer lecturas sobre la información alojada en las solicitudes HTTP transferidas y recibidas desde el servidor, como, por ejemplo, examinar los parámetros enviados en una solicitud de tipo POST, PUT o DELETE. Este paquete de Node.js se encarga de la lógica necesaria para acceder a la información de los cuerpos en las solicitudes.
- d. **Morgan:** paquete de Node.js que funciona como “Middleware” de registro de las solicitudes y errores HTTP.
- e. **Nodemon:** herramienta que se encarga de detectar cualquier cambio en el código del servidor para relanzar la ejecución de los servicios aplicando los cambios detectados.

## 2.3. Comandos y direccionamiento básico

A continuación, se especifican algunos de los comandos que se ejecutan en una terminal tipo Shell y son utilizados durante la creación y gestión de una aplicación desarrollada con Node.js junto a su función:

- A. npm init:** comando usado para configurar un paquete npm nuevo o existente. Crea el archivo package.json.
- B. npm start:** ejecuta el comando predefinido en la propiedad “start” del objeto Script de un paquete (package.json).
- C. npm install express:** instala el paquete de Express en el proyecto y toda aquella dependencia adicional requerida por Express.
- D. npm install mongoose:** instala el paquete de Mongoose en el proyecto y toda aquella dependencia adicional requerida por Mongoose.
- E. npm install Morgan:** instala el paquete de Morgan en el proyecto y toda aquella dependencia adicional requerida por Morgan.
- F. npm install body-parser:** instala el paquete de Body-parser en el proyecto y toda aquella dependencia adicional requerida por Body-parser.
- G. npm install nodemon:** instala el paquete de Nodemon en el proyecto y toda aquella dependencia adicional requerida por Nodemon. Es necesario indicar el uso de Nodemon en el script start del paquete Json.

Recuerde que las Rutas o “Routes” son el concepto usado en Node.js para referirse a los componentes que permiten gestionar el direccionamiento dentro de una aplicación. Estas permiten asociar los métodos HTTP (GET, POST, PUT, DELETE, etc.) un

patrón o camino URL (URL path) y con la función que será ejecutada para controlar dicho patrón.

En Node.js se utiliza el paquete de Express para la gestión de las rutas y tendrá una estructura similar a la que se visualiza en la siguiente figura:

**Figura 4.** Esquema general de manejo de rutas en Node.

```
1  const express = require('express')
2  const router = express.Router()
3
4  const controllerA = require('../controllers/ControllerModel_A')
5
6  router.get('/', controllerA.logic)
7  router.post('/path', controllerA.logic_2)
8  router.post('/path2', controllerA.logic_3)
9
10 module.exports = router
```

```
1 const express = require('express')
2 const router = express.Router()
3
4 const controllerA = require('../controllers/ControllerModel_A')
5
6 router.get('/', controllerA. Logic)
7 router.post('/path', controlterA.logic_2)
8 router.post('/path2', controlLleraA.logic_3)
9
10 module.exports = router
```

Como se puede observar en la imagen se necesita de una instancia de la clase Router() de express; luego, se debe instanciar el controlador que tiene definidas las acciones para un modelo específico y finalmente, con la instancia de la clase “router” definimos el método HTTP, la ruta relativa por donde será accedido desde una solicitud web a la API publicada por el servicio junto a la función específica en el controlador encargado de darle respuesta.

### **3. Construcción API**

En este momento ya disponemos de las herramientas de “software” necesarias y los conceptos involucrados en la construcción de la API RESTful, ahora veamos los siguientes aspectos a tener en cuenta:

#### **3.1. Conexión Node.js y MongoDB**

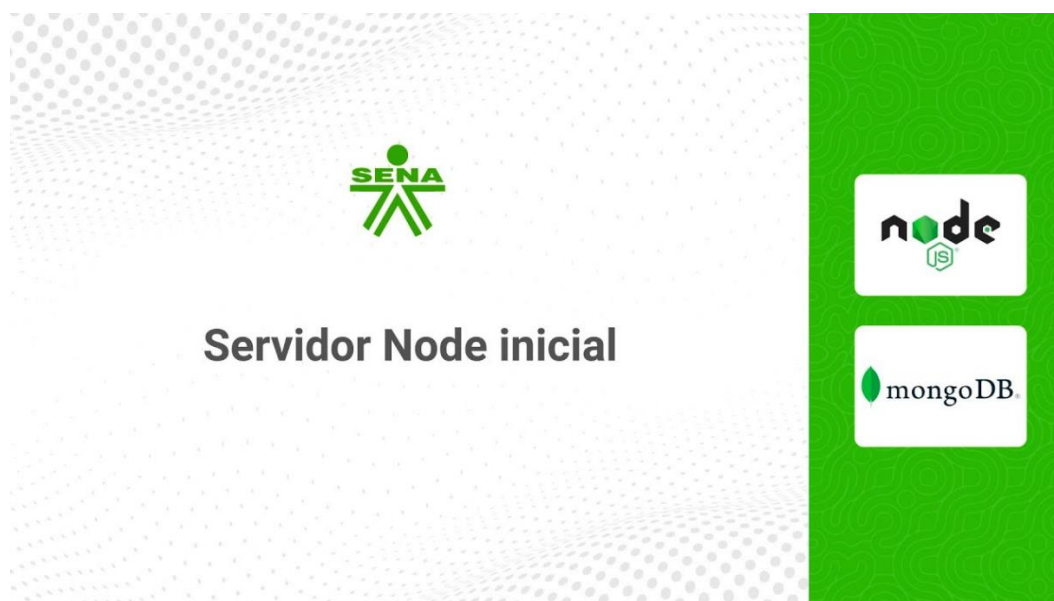
Lo primero que se debe realizar es la construcción de una aplicación Node.js que se pueda ejecutar satisfactoriamente y luego tratar de establecer una conexión con el servidor MongoDB que previamente ha sido configurado como sistema gestor de base de datos.

Se recomienda realizar los siguientes pasos:

- I. Crear una carpeta en la que se almacenará el código fuente de la aplicación.
- II. Utilizar el IDE Visual Studio Code para cargar esta carpeta e iniciar con la configuración del espacio de trabajo con la ejecución del comando `npm init`.

- III. Incluir todas las dependencias a ser usadas por el proyecto por medio del comando `npm install paquete`.
- IV. Una vez definidas las dependencias se debe crear un archivo de extensión `js` donde se importarán todas las dependencias, la lógica de conexión a la base de datos utilizando Mongoose.
- V. Finalmente, el puerto por donde el servidor escucha las solicitudes una vez se ejecuta el comando `npm start`.

### Video 3. Servidor Node.js



[Enlace de reproducción del video](#)

#### Síntesis del video: Servidor Node.js

Hola estimado aprendiz, ahorita vamos a crear nuestro primer servidor Node.js utilizando la herramienta Visual Studio Code, para eso vamos a ir a nuestro equipo y



en el sistema de archivos de nuestro equipo local vamos a crear una carpeta en este caso yo cree la carpeta “Backennode” es una carpeta vacía, voy a volver aquí Visual Studio Code y voy a abrir ese “folder” ya que quiero que quede almacenado todo lo que yo haga en ese “folder”.

Entonces busco mi carpeta le digo abrir. Bien entonces lo que vamos a hacer ahorita es una vez abra este espacio vamos a crear un archivo, voy a decirle que si confío en el autor de esta carpeta para que me lo deje cargar, voy a cerrar aquí unas advertencias; entonces lo que voy a hacer es crear dentro de esa carpeta ya habiéndola cargado voy a crear un nuevo archivo yendo a la parte superior izquierda en la opción “new file” y voy a crear un archivo llamado server.js, el punto js le indica que es un archivo de JavaScript que es lo que va interpretar nuestro servidor y vamos a hacer uso de las librerías de Node.js, para eso entonces voy a ir aquí a mi terminal voy a crear una nueva terminal para poder ejecutar unos comandos, aquí tengo mis comandos y vamos a iniciar con los comandos que me permiten inicializar todo el proyecto, todos los elementos que necesito para utilizar Node.js, entonces lo primero que tengo que hacer es poner el comando npm init para inicializar el proyecto, aquí me va a pedir información con la que va a quedar configurado el proyecto, entonces nombre del paquete, aquí pueden colocar el nombre que quieran voy a colocarle apptest es importante que el nombre inicie con minúscula no puede tener ningún carácter en mayúscula, bueno sino quiero configurar esto simplemente le doy enter para que me deje los valores por defecto y voy a colocar solamente el nombre del autor.

Luego la configuración que básicamente es un archivo de extensión punto json donde está configurada toda esta información, le voy a indicar que sí que termine esa configuración.

Todo esto que yo acabo de hacer aquí por consola del terminal va a quedar almacenado en este archivo package.json yo puedo editarlo en el momento en que lo necesite.

Bueno en segunda instancia la idea y la potencia del Node.js es que yo puedo utilizar un conjunto de dependencias que me van a facilitar el trabajo, entonces voy a instalar esas dependencias entonces lo voy a hacer a través del comando npm voy a decirle que instale varias dependencias que va a utilizar, en primer lugar express que es básicamente un framework de Node.js que me permite hacer muy rápidamente aplicaciones web y hay servicios de tipo API; voy a instalar mongoose que es el que me va a permitir conectarme fácilmente con MongoDB, voy a instalar Morgan que en esencia Morgan lo que me hace es permitirme hacer tipos de seguimiento y traseo a las solicitudes http y body-parser para parsear la respuesta que recibo desde el servidor: bien entonces una vez finaliza de instalar esas dependencias ya tendríamos los elementos listos para poder iniciar nuestro primer servidor Node.js con una conexión exitosa a MongoDB. Para eso vamos a utilizar la cuenta y el usuario de nuestra base de datos que ya habíamos configurado anteriormente, bueno ahí termino el proceso.

Entonces recordemos que anteriormente ya teníamos configurado un clúster, habíamos definido unos usuarios, es muy importante verificar que tenga un usuario que yo pueda utilizar cierto, recordemos que estas credenciales de este usuario es las

que vamos a utilizar para conectarnos. En segunda instancia que yo haya configurado la IP donde estoy actualmente trabajando para que esté habilitada y este activa, yo he verificado mi dirección IP, coincide con la que tengo en este momento entonces no habría ningún problema.

Si, generalmente los problemas son porque no estoy trabajando con el usuario que definí o simplemente mi IP está bloqueada; también muy importante tener en cuenta que las credenciales tratar de no utilizar credenciales como @ o que finalice en @ o con numeral ya que igual vamos a crear una cadena de conexión y esas cadenas de conexión hay veces esos caracteres especiales me genera problemas.

Bien, volviendo acá a nuestro servidor vamos a empezar ahora a codificar, entonces lo primero que tengo de hacer es empezar a importar todas las librerías que necesito en primer lugar express, necesito express entonces voy a decirle aquí que requiero utilizar express, ahí está, de la misma forma voy a utilizar las demás, entonces mongoose, entonces requiero utilizar mongoose, Morgan, y body-Parser, listo entonces ya importó las dependencias que yo ya instalé en mi proyecto y entonces vamos a empezar con la conexión, entonces en primera instancia vamos a utilizar esta instancia de mongoose que ya importamos, vamos a utilizar la función de conectar y aquí por parámetro le voy a indicar la cadena de conexión. La cadena de conexión yo la obtengo directamente desde Mongo, entonces si vengo acá a Database acá yo puedo por ejemplo buscar las colecciones que tengo; yo en el proceso instale la base de datos que venía de pruebas, pero yo puedo crear una base de datos si quiero voy a llamarla "test" y debo crear una colección inicial, aunque este vacía, voy a decirle

créate, entonces me voy a conectar a esa base de datos “test” que tengo acá y a través de ella voy a hacer todo el proceso de gestión.

Pero entonces lo que me interesa realmente aquí de database es que ahí hay una opción que me dice “coonect” , cuando yo le digo “coonect” me saca las opciones de cómo me puedo conectar, entonces yo voy a conectarme a través de una aplicación entonces voy a indicarle aquí “Connect you application”, acá me indica que necesito Nod.js pues que ya lo tenemos instalado en nuestra máquina y aquí me saca una cadena de conexión genérica, me advierte que aquí yo ya estoy utilizando el usuario que ya está establecido que yo lo puedo modificar si quiero, el “password” aquí debo indicar el “password” de esa credencial específica y además de eso me coloca aquí MyFirsteDatabase pero yo puedo cambiarlo con el nombre de la base de datos que yo necesito; Entonces voy a copiarlo, vengo acá a la cadena de conexión y la pego, bueno voy a crearme aquí otra variable simplemente para manejar el “password”, como esto es de prueba sencillamente voy a colocar un “password” muy sencillo que ya definí y lo voy a utilizar acá en vez de este “password” voy a indicarle que el “password” es este, y aquí la base de datos no va a utilizar MyFirstDatabase sino la base de datos “test” que acabamos de crear. Listo eso sería todo.

Ahora voy a crear aquí una variable va a hacer mi base de datos a la cual le voy a asignar la condición de Mongose que ya está definido con esa cadena de conexión anteriormente descrita Connection y entonces para hacer seguimiento de que esa conexión fue exitosa o no fue exitosa pue solo que vamos a hacer es (voy a ampliar un poquito aquí más la letra para que lo podamos ver más claramente, eso mucho mejor). Entonces en este objeto que es el de la conexión vamos a definirle unos

listener para mirar si el proceso fue correcto, entonces a través de método y vamos a definir un listener simplemente voy a colocarle el nombre del evento que voy a seguir, entonces vamos a verificar si en algún momento hay un error.

Si hay un error entonces vamos a recibir ese parámetro, una función anónima que estamos creando acá y esta función anónima lo que va a hacer es escribirme en pantalla a través de la consola con un log un mensaje que me advertirá que ha pasado, vamos a indicarle aquí que error en la conexión a la base de datos, muy bien. Y voy a definir un segundo listener que se guardara solamente una vez y este es solamente cuando la cadena de conexión y la conexión haya sido exitosa con el servidor, entonces cuando ocurra el evento open ó sea fue exitoso, vamos a crear aquí una función anónima otra vez sin parámetros sencillamente le vamos a indicar aquí también de la misma forma vamos a hacer por la consola un mensaje que me indique que conexión exitosa. Muy entonces ya tenemos definidos estos elementos, ya tenemos definido la conexión como tal y controlar a través de listener si se conecta exitosamente o no, entonces ahora si vamos a crear los elementos propios del servidor, reutilizando todos esos componentes que ya están definidos, entonces vamos a crear nuestra “app”, la “app” va a ser una instancia de “express” y vamos a usar en esa instancia de express todas esas librerías que están definidas en la parte inicial, entonces vamos a usar Morgan, vamos a usar el body-Parser básicamente lo que estamos definiendo acá es que como vamos a recibir una respuesta a través de “http” cierto unas solicitudes al servidor, entonces necesito de cierta forma establecer esa información, poder leerla fácilmente, entonces si el mensaje viene codificado o si viene en otro formato diferente pues estas librerías me van a ayudar a poder leerlo.

Entonces vamos a decirle aquí que utilice el body-Parser y también la respuesta puede ser procesada a través del body-Parser pero en formato json.

Bien entonces ya estamos casi listos, entonces ya tengo mi instancia de express estoy utilizando las dependencias, ya tengo mi conexión cierto establecida; finalmente voy a definir aquí una variable para definir el puerto en el cual va a correr el servidor, entonces voy a indicarle acá. Vamos a mirar primero si en el archivo de ambiente vamos a tener definido un puerto por defecto, si está definido por defecto pues vamos a utilizar ese puerto y si no vamos a utilizar el puerto 3.000, ahora si vamos a correr nuestra aplicación le vamos a decir que arranque escuchando por este puerto definido y vamos a decirle acá que cuando arranque nos escriba que se inició correctamente, voy a indicarle aquí servidor corriendo en el puerto, aquí vamos a pegarle el PORT para probar.

Entonces vamos a volver aquí a la terminal y vamos a ejecutar el comando de npm start para que arranque nuestro servidor, y vamos a ver que va a pasar.

Bueno este es un error muy común no pasó nada en teoría de ver acá que se conectó no se conectó y que si está corriendo. Recuerden siempre que cuando escriban código tienen que salvarlo, grabarlo listo ya lo grabe entonces ahora si vuelvo a correr el comando npm start y ahí me dice que el servidor está corriendo y que la conexión fue exitosa, es decir en este momento estamos directamente a nuestro servidor MongoDB que es lo que estamos estableciendo en la conexión con ese usuario y las credenciales definidas y estamos conectados a la base de datos test que es la base de datos que tenemos acá en nuestro servidor, es decir, todo está listo para trabajar con la base de datos test.

En principio no tenemos nada la idea es desarrollar todos los elementos, las colecciones, los documentos para poder manipularlo eso lo haremos más adelante, pero lo importante era establecer la conexión.

Finalmente, cada vez que yo haga un cambio acá recordemos que yo hago un cambio acá error a la base de datos por ejemplo de MongoDB, acordémonos siempre de salvar, acá esto queda corriendo la terminal, le damos control C para terminar, nos toca grabar recordemos siempre grabar y volver a correr.

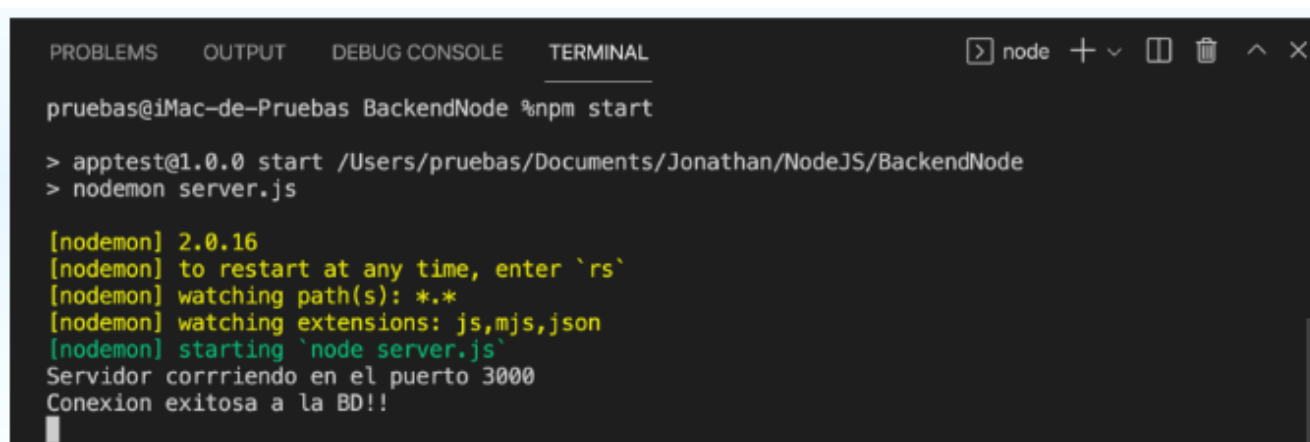
Hay otra opción que es utilizar una dependencia muy útil que es: nodemon entonces voy a romper esto voy a decirle npm install nodemon, es muy útil eventualmente cuando estemos trabajando con el servidor necesitamos hacer todo el tiempo modificaciones en el archivo fuente y muy probablemente necesite y se me olvide grabar y volver a correr y no este corriendo los cambios entonces esta me va a servir.

Entonces vuelvo aquí a package.json y aquí donde es scripts se le escribe start el está arrancando node server.js que es nuestro archivo de servidor voy a decirle nodemon, voy a grabar, listo y entonces que pasa. Voy a correr nuevamente mi servicio npm start arrancó, la diferencia radica en que si yo hago cambios por ejemplo aquí yo le digo conexión exitosa a la BD y grabo automáticamente la nodemon viene a la terminal me ejecuta los cambios y me vuelve a recompilar el servidor para que quede funcionando nuevamente.

Con esto tenemos nuestro primer servidor conectado a una base de datos MongoDB exitosamente.

Basado en la explicación anterior, en la siguiente figura se visualiza el resultado final esperado luego de realizar el proceso de creación y configuración inicial del servidor:

**Figura 5.** Ejecución del servidor Node.js por medio de la terminal de “Visual Studio Code”



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
pruebas@iMac-de-Pruebas BackendNode %npm start

> apptest@1.0.0 start /Users/pruebas/Documents/Jonathan/NodeJS/BackendNode
> nodemon server.js

[nodemon] 2.0.16
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Servidor corriendo en el puerto 3000
Conexion exitosa a la BD!!

```

PROBLEMS OUTPUT DEBUG CONSOLE – TERMINAL

pruebase@iMac-de-Pruebas BackendNode @nnp start

```

> apptest@1.0.0 start
/User/pruebas/Documents/Jonathan/NodeJS/BackendNode
> nodemon server.js

```

```

[nodemon] 2.0.16
[nodemon] to restart at any time, enter "rs"
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting *node server.js
Servidor corriendo en el puerto 3000
Conexion exitosa a la BD!!

```



### 3.2. Construcción de consultas con Node

Para la construcción del servicio RESTful se recomienda utilizar algún patrón arquitectónico que facilite la disminución de dependencias, la actualización rápida de componentes y facilidades para el escalamiento y pruebas. Tenga en cuenta que esta API es la responsable de definir el esquema de base de datos y la forma en que será manipulado por los clientes.

Para controlar el esquema de la base de datos deberá hacer uso de una instancia de la clase Schema de la librería Mongoose.js, con el cual definirá un modelo o estructura general de los documentos que serán almacenados en MongoDB, especificando como mínimo cada uno de los campos con su correspondiente tipo de dato.

Para tener más detalles puede consultar el siguiente enlace:

<https://mongoosejs.com/docs/guide.html>

Revisemos entonces a través de la siguiente figura la estructura general requerida para crear un nuevo modelo:

**Figura 6.** Esquema general de un modelo.

```

1  const mongoose = require('mongoose')
2  const Schema = mongoose.Schema
3
4  const newSchema = new Schema({
5    campo1: {
6      type: String
7    },
8    campo2: {
9      type: Number
10   }
11  }, {timestamps : true})
12
13  const nombreDoc = mongoose.model('nombreDoc',newSchema)
14  module.exports = nombreDoc

```

```

1  const mongoose = require('mongoose')
2  const Schema = mongoose.Schema
3
4  const newSchema = new Schema({
5    campo1: {
6      type: String
7    },
8    campo2: {
9      type: Number
10   }
11  }, {timestamps : true})
12
13  const nombreDoc = mongoose.model('nombreDoc',newSchema)
14  module.exports = nombreDoc

```

Una vez definidos los modelos se deben crear los controladores que determinan las operaciones a realizar en dicho modelo, entre las más comunes se encuentran:

**Controlador C.** Buscar un documento específico.

**Controlador D.** Listar todos los documentos.

**Controlador E.** Registrar un nuevo documento, etc.

Los controladores harán uso de los métodos que serán emparejados con los comandos que se ejecutan del lado de MongoDB.

Si desea profundizar en las funciones disponibles para los modelos de Mongoose, consulte la documentación oficial disponible en:

<https://mongoosejs.com/docs/models.html>

Para ejemplificar un poco este tema, en el siguiente video se presenta un ejemplo, que muestra el proceso para crear un modelo y su correspondiente controlador, con lo cual se responde a tres diferentes funciones asociadas a los métodos HTTP GET y POST:

## Video 4. Modelo y controlador



[Enlace de reproducción del video](#)

### Síntesis del video: Modelo y controlador

Muy bien, en este punto ya tenemos listo nuestro servidor node.js corriendo exitosamente por el puerto 3000 con una conexión directa a nuestra base datos MongoDB, ahora vamos a terminar de construir los elementos que nos faltan para nuestro API RESTfull.

Entonces lo primero que tenemos que hacer es que dentro de la carpeta vamos a crear unos paquetes, entonces vamos a crear unas carpetas para estructurar nuestro proyecto en tres partes: una primera parte vamos a manejar los modelos, en otra parte vamos a manejar los controladores y en una tercera carpeta vamos a

manejar lo que tiene que ver con las rutas, bien entonces que va a hacer la carpeta modelos. En la carpeta modelos vamos a crear nuestros modelos que van a representar la información que vamos a almacenar en la base de datos, entonces aquí voy a crearme para este ejemplo un modelo de empleado, entonces vamos a guardar, almacenar en nuestra base de datos MongoDB empleado. Recordemos que la administración y gestión del esquema y de la información que se va a almacenar se hace directamente en la API; en esta carpeta modelo vamos a definir los esquemas que vamos a utilizar entonces para adquirir un esquema necesitamos primero instanciar mongoose vamos a utilizar esta dependencia en mongoose y con esta dependencia definida vamos a crear entonces ahora un esquema; este esquema lo obtenemos a partir de mongoose. esquema, y entonces vamos a definir con esta variable de esquema vamos a definir nuestro empleado esquema, entonces aquí en el empleado esquema vamos a definir uno nuevo de tipo esquema y aquí en su interior vamos a especificar como va a estar estructurado este esquema, entonces lo que vamos a especificar acá es la estructura del esquema de un empleado, entonces en primer lugar vamos a decir que los empleados tienen un nombre, este nombre va a tener asociado un tipo, que el tipo va a ser String; de la misma forma definimos los otros campos que hacen parte del esquema de empleados, entonces vamos a definir también un correo electrónico, vamos a definir un celular y finalmente una edad, la edad si podría ser tipo numérico.

Entonces habiendo definido esto ya tenemos los cuatro campos, esto al final va a transformarse en una colección donde esos documentos de empleados van a tener esta estructura y adicional a eso es muy común que en las bases de datos manejemos opciones que nos permitan llevar un tipo de registro de los lots de la creación y

actualización de cada uno de los documentos que vamos insertando; entonces dentro de mongoose tenemos la posibilidad de definir el timestamps, le decimos true esto nos permite a nosotros que además de estos campos se van a agregar dos campos adicionales uno llamado createdAt y updatedAt donde se va a guardar la fecha exacta de cuando se creó y cuando se actualizó ese registro.

Listo el esquema entonces vamos a definir que entonces este empleado va a ser un modelo que vamos a llamar empleado y su esquema es el que se definió hace un momento, empleado esquema. Muy bien y lo exportamos demos module.exports empleado, listo ya nos quedó listo el esquema de empleado.

Ahora vamos a irnos a los controladores, en los controladores lo que hacemos es que vamos a definir que operaciones vamos a poder hacer sobre este esquema de empleado; entonces aquí en los controladores vamos a crear un archivito ese archivito de controladores voy a llamarlo: empleado controller.js. este empleado controller vamos a definirle aquí que vamos a utilizar empleado que acabamos de definir entonces vamos a decirle aquí la dirección relativa está en models y entre el models empleado.

Bien entonces aquí lo que vamos a hacer en controladores es definir cuáles son las acciones que se podrán utilizar, entonces para este ejemplo vamos a crear tres acciones, vamos a poder consultar la lista de empleados, todos los empleados entonces vamos a decir aquí un "index" voy a llamarla "index" esta funcioncilla va a requerir una solicitud un request, va a tener una respuesta y vamos a invocar a una función next que es lo que vamos a hacer acá.

Entonces vamos a invocar un nivel empleado un método que es el método find, entonces este método find que estamos definiendo acá es el equivalente a ejecutar el comando find o la consulta find en Mongo; entonces aquí le vamos a decir, vamos a tratar de controlar las respuestas, cuando tenga una respuesta vamos a decirle que esa respuesta formato json va a ser de respons, ósea la respuesta que me llegue es la que yo voy a enviar como respuesta cuando se haga la solicitud.

Ahora esto puede tener un problema que puede capturar cualquier error que pueda ocurrir, entonces vamos a controlar aquí el error en caso de que ocurra un error lo que vamos a hacer es que vamos a poner en la respuesta en formato json un valor de mensaje entonces lo estamos personalizando, que vamos a decir “ocurrió un error”, está listo.

Entonces una vez se consulta esta función “index” él va a trabajar con la respuesta obtenida desde el servicio si la respuesta llega bien simplemente vamos a retornar en formato json esa misma respuesta, pero si llega a haber un error vamos a predefinir un mensaje de error como respuesta una vez se invoca al “index”.

Haríamos algo similar para consultar a todos los empleados. Vamos a cambiar el nombre de “index” a “show” puede ser o “show old”, la diferencia del anterior pues acá en este caso la idea es que vamos a hacer una consulta específica para que se diferencie del anterior. ¿Entonces a diferencia del anterior necesitamos que nos envíen un parámetro que va a ser el código del empleado, ese código del empleado de donde lo vamos a sacar? Cuando se envíe la solicitud en request en el cuerpo de esa solicitud me deben enviar a mí el código de ese empleado entonces yo lo voy a capturar acá y lo voy a utilizar para invocar al método findByDy lo voy a pasar por

parámetro ese código de empleado, este método igual va a procesar una respuesta, si la respuesta está bien vamos a mandar ese mismo response y vamos a capturar el error; si se captura el error vamos a colocar el mensaje personalizado de “ocurrió un error”.

¿Entonces ya tenemos el “index” que me muestra todo, el show y vamos a hacer una tercera acción que será la de ingresar un empleado, entonces como hago para ingresar? Vamos a llamar a esta guardar, bueno para guardar un empleado necesito mucho más que todo esto necesito que me envíen un elemento que represente el esquema del empleado que ya creamos anteriormente; recordemos que el empleado en su esquema tiene cuatro campos y dos que se generan de forma automática. Entonces voy a crear acá un empleado, este empleado va a estar estructurado por la siguiente información: va a tener un nombre, este nombre lo debes recibir en la solicitud en el request, body. nombre lo mismo el correo, el correo también lo debes solicitar en el request en el body. Correo cada uno de los elementos separados por coma excepto el último y celular y la edad.

Listo una vez tengamos esto, entonces ahora lo que vamos a hacer es invocar a empleado punto save (empleado.save) y le vamos a pasar este mismo objeto de empleado que tengo acá es el que voy a utilizar acá, este empleado ya tiene estructurado internamente toda la información, entonces vamos a invocar el método save y de igual forma si, en este caso pudo salvar pues no voy a enviar esta información sino que voy a hacer algo personalizado también voy a indicarle como esto ya sería una solicitud que se guarda en el servidor como tal si se hace exitosamente voy a decirle empleado registrado y si hay algún tipo de error “ocurrió un error” listo entonces finalmente acá le voy a decir el módulo que exporte, esto



que ya tenemos acá entonces vamos a exportar estos elementos que ya tenemos acá que son “index”, “show” y guardar, deben de quedar: “index”, “show” y guardar entonces ya me quedó listo mi esquema listos los controladores y lo único que me falta por construir son las rutas.

El último paso para completar la API RESTful es la construcción de las rutas. Tenga en cuenta que estas deben ser importadas en la clase de servidor. Se recomienda tener direcciones de rutas por cada modelo del esquema de la base de datos e implementar como mínimo en el controlador todos los métodos tipo CRUD que serán utilizados por los clientes externos.

Es por ello, que en el siguiente video se muestra un ejemplo, en el que se realiza el proceso para crear las rutas de cada uno de los métodos definidos en el controlador y la forma de adecuar el servicio para hacer uso de estas rutas; también se indica cómo utilizar la herramienta Postman para hacer las pruebas manuales de cada una de las rutas definidas.

## Video 5. Pruebas y rutas con Postman



[Enlace de reproducción del video](#)

### Síntesis del video: Pruebas y rutas con Postman

Bien, ahora nos dedicaremos a la última parte para la construcción de la API, nuestro backend de empleado y es construir las rutas. Entonces en la carpeta de rutas vamos a crear un archivo que voy a llamar empleado.js aquí voy a especificar las rutas disponibles para lo que tiene que ver todo con el esquema de empleados, entonces voy a utilizar “express” ya que el me gestiona las rutas, vamos a importarlo, vamos a crear entonces ahora nuestro “Router” que será una instancia de “express Router” y vamos a definir entonces que necesitamos utilizar el controlador de empleado para poder manejar esta ruta, vamos a crear aquí un empleado controller el cual se importará desde el controlador que definimos anteriormente que está ubicado en controllers empleado controler; muy bien entonces vamos a definir nuestras rutas, entonces con nuestro “Router” vamos a definir que hay un “get”

una ruta para get que vamos a resolver a través del controlador, aquí creamos un método llamado index. Esto es un get porque lo que hace es consultar todos los empleados y no necesita ningún parámetro.

Vamos a crear otra ruta de tipo post, que esta la vamos a ubicar en show y lo va a resolver nuestro controlador de empleado y lo que definimos como show allá.

Y una tercera ruta que también va a ser de tipo post porque es una inserción, la ruta la vamos a encontrar aquí en guardar y se va a encargar el controlador con su elemento guardar; listo quedaron definidas las rutas exportamos esto y nos quedó listo nuestro Router de empleado.

Ahora que me falta hacer, lo que me falta hacer ahora es ir al servidor que es el que estamos ejecutando, entonces vamos a nuestro server y aquí en nuestro server donde estamos estableciendo la conexión, vamos a hacer uso de las rutas, entonces vamos a decirle que vamos a crear una ruta de empleado, empleado Router, vamos a importarla la que acabamos de definir, eso está en la carpeta de rutas y aquí la llamamos empleado; una vez definido nos conectamos y una vez establecida la conexión, empezamos a escuchar, vamos a decir que la “app” va a usar la siguiente extensión; recordemos que esto está corriendo por el local host en el puerto 3000 vamos a colocar / (slash) API / (slash) empleado tomo la dirección raíz de la ruta de empleado, entonces dentro de este local host dos puntos 3000 slash API slash empleado aquí dentro voy a usar lo que ya tenemos definido en las rutas de empleados, entonces voy a colocarle aquí empleado Route, es decir que dentro de esta dirección vamos a tener una subdivisión por cada uno de los elementos y de las funciones que definimos anteriormente en las rutas, entonces tenemos un get en la

raíz para llamar al index un slash show que va a llamar al otro método que lista por ID y el de guardar con un slash guardar. Entonces aquí ya me quedó listo esto, grabo y voy a correr nuevamente mi servidor.

Bien entonces está corriendo en el puerto se conectó exitosamente y vamos a probar ahora que si está funcionando nuestra API.

Entonces recordemos que ya estamos conectados a esta base de datos test, en este momento pues hay una colección de prueba que creamos en ese momento, pero vamos a consumirlas directamente utilizando nuestra herramienta Postman.

Entonces en primer lugar vamos a probar que realmente si podemos insertar registros para poder consultar luego; entonces acá tenemos que ubicar que es una solicitud de tipo post, habíamos dicho que eso va a estar, nuestro servicio como tal está corriendo en local host puerto 3000, y de aquí para adelante viene la ubicación de la ruta como tal.

Entonces recordemos que según lo definido acá slash api slash empleado entonces vamos a colocar /API/empleado y para el caso de lo queremos hacer que es utilizar el registrar es un slash guardar, entonces aquí /guardar, bien como esto es una solicitud post entonces necesito enviar parámetros y los vamos a enviar en el cuerpo de la solicitud, entonces recordemos que acá en el cuerpo de la solicitud, en el request vamos a tener que enviar esta información para que el pueda hacer el proceso de guardar.

Entonces volviendo aquí vamos a ir aquí en la opción que dice body, en la tercera opción, vamos a editar ese body y vamos a indicarle aquí que vamos a enviar un texto en formato json, aquí how formato json y aquí le especificamos que es lo

que le vamos a enviar, entonces el espera recibir un nombre, voy a llamarlo jhonatan, un correo no se olviden de separar por comas cada uno de los elementos, celular y finalmente la edad, esta edad es numérica voy a colocar el número directamente; listo entonces me quedo definido y ya estamos listos para hacer la solicitud, entonces tipo post y vamos a darle send y aquí me dice mensaje registrado es decir que la respuesta fue exitosa y me retorno al valor que habíamos dicho. También puedo verificarlo directamente en el servidor entonces voy a abrir aquí el servidor, voy a aquí a actualizar esto vamos a entrar aquí a colecciones, en test, miren que ya aparece la colección de empleados y en empleados está el empleado que acabé de registrar con sus cuatro campos y estos son los últimos dos campos que comentaba que se creaban de forma automática de cuando fue creado el registro y cuando se actualizaba, en un formato time stand, de la misma forma puedo crear un segundo registro de empleado, registrado si actualizamos acá ya tenemos nuestro segundo, es decir que nuestra API en lo que corresponde a las solicitudes de tipo post de guardar están funcionando.

Vamos a probar las otras dos que creamos, entonces voy a abrir una pestaña nueva voy a crear una solicitud que voy a copiar esto, entonces habíamos visto que si yo hacía una solicitud get en la raíz, debería contenerme todos los empleados registrados; voy a enviar la solicitud y aquí tengo la respuesta de todos los empleados registrados hasta el momento y también habíamos probado otra solicitud muy parecida a esta pero era de tipo post para buscar un empleado en particular, entonces voy a marcarle aquí post, un empleado particular, ese empleado particular para poder accederlo recordemos que habíamos definido unas rutas, entonces para el caso de ese señor es la ruta “show”, entonces vamos con la ruta “show” y tengo

que pasarle un parámetro, que este parámetro será también en formato json y debe ser un parámetro de tipo según esto en el controlador habíamos definido que ese parámetro iba a ser un empleado ID, entonces vamos a colocarle aquí que este empleado que vamos a buscar es un empleado particular, entonces vamos a buscar por ejemplo el segundo registro el ID. Otra cosa interesante de Mongo es que el ID los genera de forma automática, entonces vamos a buscar este empleado particular llamado este, y aquí lo encontró. ¿Qué pasa si yo envío un código de alguien que no existe? Probémoslo. Entonces me dice ocurrió un error que fue lo que definimos específicamente en el controlador y con esto tenemos una prueba de que tenemos ya nuestra API por lo menos con lo que corresponde a métodos “get” y métodos post definidos; con esto tenemos ya los instrumentos para construir los elementos que faltarían para completar el “croud” que sería hacer unos “deit” y un “delete”.

En el ejemplo anterior solo se implementaron las solicitudes HTTP de tipo GET y POST, la implementación de los otros métodos HTTP que complementan el CRUD siguen la misma dinámica; por ende, con los conocimientos adquiridos es posible completar el ejercicio sin ningún problema.

Recordemos que las operaciones que se pueden realizar sobre los documentos están establecidas en la clase controlador que importa el modelo particular sobre el que se realizarán las opciones. Así, se presentan algunas de estas funciones de Mongoose que pueden ejecutar acciones del lado de la base de datos MongoDB, conozcámosla:

#### **A. findByIdAndUpdate().**

Recibe como parámetro el Id del documento a buscar para la actualización y el objeto con la información a ser utilizada en la actualización.

Si encuentra el registro, le aplica los cambios determinados en el objeto enviado y retorna el registro encontrado como respuesta.

Ejecuta el comando findOneAndUpdate en MongoDB

#### **B. findByIdAndDelete()**

Recibe como parámetro el Id del documento a buscar, que será eliminado.

Ejecuta el comando findOneAndDelete() en MongoDB

#### **C. Save()**

Inserta un nuevo documento a la colección.

Debe crear un objeto del modelo que quiere registrar, llenar los campos e invocar a la función save()

Esta función retorna un valor indefinido, si hay error o el documento registrado, si el proceso es exitoso.

#### **D. find()**

Recibe como parámetro un filtro que se utiliza para encontrar documentos en el esquema. Si el parámetro recibido es un filtro vacío, quiere decir que se buscan todos los documentos del esquema.

Algunos ejemplos de la utilización del find podrían ser los siguientes:

```
//Encuentra todos los documentos de la colección MyModel  
MyModel.find({})  
  
//Encuentra todos los documentos de la colección MyModel cuyo campo  
name sea igual a 'john' y la edad sea igual o superior a 18.  
MyModel.find({ name: 'john', age: { $gte: 18 } })
```

### E. **findById()**

Recibe como parámetro el Id del documento a buscar en la colección.  
Retorna como resultado un único documento si es que dicho Id existe.  
Este método es equivalente al comando `findOne()` de MongoDB, en el que solo se busca por el campo de identificación del documento.

En la siguiente figura se muestra un borrador de la estructura que tendría una función en el controlador para actualizar una colección de tipo Modelo X, se puede observar que en el controlador se debe importar primero el modelo para poder invocar la función `findByIdAndUpdate()`, la cual recibirá como parámetro el Id del documento a actualizar y el documento con toda su estructura de cómo debería quedar luego de la actualización. El método `findByIdAndUpdate()` asocia a la palabra reservada `$set` los valores a actualizar. Tenga en cuenta que cualquier campo de la colección puede ser actualizado excepto el identificador.

Los parámetros utilizados en la consulta son obtenidos desde el “body” del objeto de tipo “request”, es decir, que en el momento de hacer la solicitud del servicio



se deberá hacer uso de un método de tipo POST o similar que acepte el envío de parámetros. revise dicha figura:

**Figura 7.** Esquema general para la actualización de un documento de un modelo específico.

```
const Modelo = require('../models/ModeloX')

const update = (req, res, next) => {
  let modelID = req.body.modelID
  let modelUpdateData = {
    campo1: req.body.campo1,
    campo2: req.body.campo2,
    campo3: req.body.campo3,
    campo4: req.body.campo24
  }
  Modelo.findByIdAndUpdate(modelID, {$set: updateData})
    .then(response => { res.json({ response }) })
    .catch(error => { res.json({ message: 'Ocurrio un error' }) })
}
```

```
const Modelo = require('../models/ModeloX')

const update = (req, res, next) => {
  let modelID = req.body.modelID
  let modelUpdateData = {
    campo1: req.body.campo1,
    campo2: req.body.campo2,
    campo3: req.body.campo3,
    campo4: req.body.campo24
  }
  Modelo. findByIdAndUpdate(modelID, {$set: updateData})
    .then(response => { res.json({ response }) })
    .catch(error => { res.json({ message: 'Ocurrio un error' }) })
}
```

Por último, en la siguiente figura, se muestra un borrador de la estructura de una función que permite la eliminación de un documento, de acuerdo con un número de identificador dado. Al igual que las demás operaciones vistas, se requiere la importación del modelo sobre el que se va a realizar el proceso de eliminación y la invocación de la función `findByIdAndRemove()`

El parámetro del identificador debe ser capturado desde el “body” del objeto “request”, lo que implica que la invocación de este servicio se realizará por medio de un método POST o similar que acepte el envío de parámetros en el cuerpo de la solicitud enviada al servidor. Dicha figura es:

**Figura 8.** Esquema general para la eliminación de un documento de un modelo específico.

```
const Modelo = require('../models/ModeloX')

const destroy = (req, res, next) => {
  let modelID = req.body.modelID
  Modelo.findByIdAndRemove(modelID)
    .then(response => { res.json({ response }) })
    .catch(error => { res.json({ message: 'Ocurrio un error' }) })
}
```

```
const Modelo = require('../models/ModeloX')

const destroy = (req, res, next) => {
  let modelID = req.body.modelID
  Modelo.findByIdAndRemove(modelID)
    .then(response => { res.json({ response }) })
    .catch(error => { res.json({ message: 'Ocurrio un error' }) })
}
```

Finalmente, recuerde que cada función del controlador debe ser explotada al módulo por medio del comando:

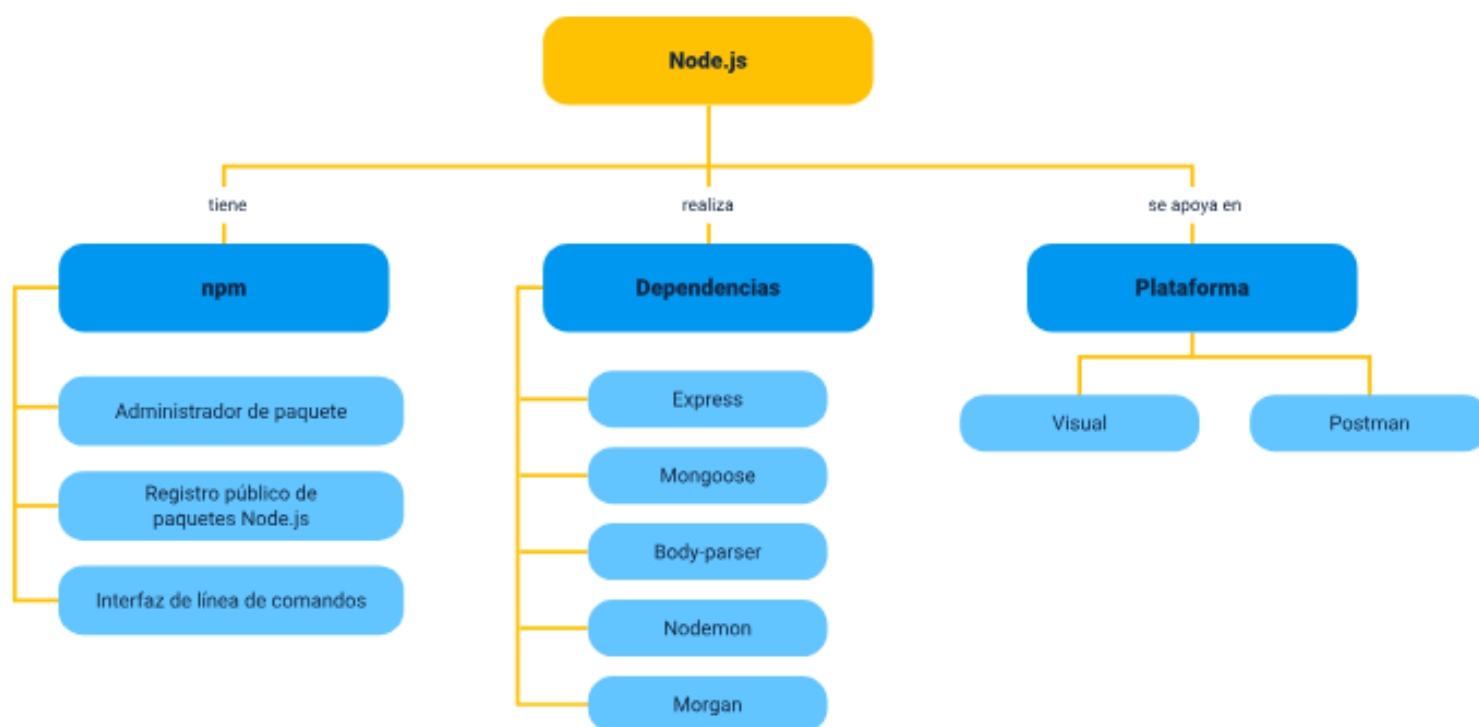
**`module.exports = {fun1, fun2, ... fun n}`**

En el que fun1, fun2 y fun n corresponden a las funciones implementadas para la gestión de operaciones sobre los modelos construidos y que, adicionalmente, cada una de estas funciones deberán tener su respectiva representación en el archivo de rutas.

Se recomienda usar las indicaciones del video “Pruebas y rutas con Postman” para completar el ejercicio de habilitar las operaciones de actualización y eliminación de colecciones y así completar todas las operaciones CRUD del Backend de ejemplo.

## Síntesis

El Node.js se considera un entorno de código abierto para JavaScript; asíncrono, diseñado para crear aplicaciones “network” escalables, por lo que se hace necesario conocer a profundidad sus generalidades, dependencias, comandos, herramientas y direccionamiento básico, entre otras cosas. De igual manera, se debe conocer de construcción API con el fin de poder crear una API RESTful que logre una comunicación con MongoDB. Por esta razón, este componente formativo profundizó estos temas, los cuales se resumen a través del siguiente esquema:



## Material complementario

Tema	Referencia	Tipo de material	Enlace del recurso
2.3. Comandos y direccionamiento básico	© OpenJS Foundation. (s. f.). <i>Index   Node.js v18.1.0 Documentation</i> .	Página web	<a href="https://nodejs.org/api/">https://nodejs.org/api/</a>

## Glosario

**Middleware:** en Node.js, hace referencia a un bloque de código que es ejecutado en el medio desde que se hace una solicitud hasta que llega al servidor.

**ORM:** modelo de programación que consiste en mapear tablas de un modelo relacional a objetos en el paradigma de programación orientada a objetos.

**Shell:** ventana de comandos.

## Referencias bibliográficas

npm, Inc. (s.f.). *npm*. <https://www.npmjs.Com>

Red Hat, Inc. (2020,). *¿Qué es una API de REST?*  
<https://www.redhat.com/es/topics/api/what-is-a-rest-api>

Stack Overflow. (2021a). *Stack Overflow Developer Survey 2021*.  
<https://insights.stackoverflow.com/survey/2021#most-popular-technologies-language-prof>.

## Créditos

Nombre	Cargo	Regional y Centro de Formación
Claudia Patricia Aristizábal	Líder del equipo	Dirección General
Liliana Victoria Morales Gualdrón	Responsable de línea de producción	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Jonathan Guerrero Astaiza	Experto temático	Centro de Teleinformática y Producción Industrial - Regional Cauca
María Fernanda Chacón Castro	Diseñadora instruccional	Centro de Gestión Industrial - Regional Distrito Capital
Andrés Felipe Velandia Espitia	Asesor metodológico	Centro de Diseño y Metrología - Regional Distrito Capital
Rafael Neftalí Lizcano Reyes	Responsable equipo desarrollo curricular	Centro Industrial del Diseño y la Manufactura - Regional Santander
Sandra Patricia Hoyos Sepúlveda	Corrector de estilo	Centro de Diseño y Metrología - Regional Distrito Capital
Alix Cecilia Chinchilla Rueda	Metodología para la formación virtual	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Yuly Andrea Rey Quiñonez	Diseñador web	Centro de Gestión de Mercados, Logística y Tecnologías de la



Nombre	Cargo	Regional y Centro de Formación
		Información - Regional Distrito Capital
Jhon Jairo Urueta Álvarez	Desarrollador fullstack	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Ernesto Navarro Jaimes	Animación y producción audiovisual	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Lady Adriana Ariza Luque	Animación y producción audiovisual	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Laura Gisselle Murcia Pardo	Animación y producción audiovisual	Centro de Gestión de Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Carolina Coca Salazar	Evaluación de contenidos inclusivos y accesibles	Centro de Gestión De Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Lina Marcela Pérez Manchego	Validación de recursos educativos	Centro de Gestión De Mercados, Logística y Tecnologías de la Información - Regional Distrito Capital
Leyson Fabián Castaño Pérez	Validación y vinculación al LMS	Centro de Gestión De Mercados, Logística y Tecnologías de la

Nombre	Cargo	Regional y Centro de Formación
		Información - Regional Distrito Capital