**M3S18: APIs con ExpressJS: CRUD Básico**

**¡Hola, bienvenid@ al Módulo 3!, ¿qué tal?, ¿cómo te sientes?, ¡que rápido pasa el tiempo, hace poco estabamos iniciando el módulo 1 y ya hasta proyectos entregaste!, me alegra muchísimo que estemos nuevamente junt@s.**

**A continuación, seguimos con el contenido correspondiente a la semana 18, en donde revisaremos APIs con ExpressJS.**

**Iniciemos… 😃**

**ÍNDICE**

* [APIs](https://github.com/U-Camp/BOOT-M3-SEM18#apis)
* [Postman](https://github.com/U-Camp/BOOT-M3-SEM18#postman)
* [ExpressJS](https://github.com/U-Camp/BOOT-M3-SEM18#expressjs)
  + [Estableciendo nuestro ambiente](https://github.com/U-Camp/BOOT-M3-SEM18#estableciendo-nuestro-ambiente)
  + [Leer elementos](https://github.com/U-Camp/BOOT-M3-SEM18#leer-elementos)
  + [Crear elemento](https://github.com/U-Camp/BOOT-M3-SEM18#crear-elemento)
  + [Editar elementos](https://github.com/U-Camp/BOOT-M3-SEM18#editar-elementos)
  + [Borrar elementos](https://github.com/U-Camp/BOOT-M3-SEM18#borrar-elementos)

**APIs**

En esta sección, hablaremos sobre la creación de nuestras APIs.

Cuando nos comunicamos con un servidor a través de una petición, este nos dará una respuesta que incluye todos los datos solicitados.

Por ello, necesitaremos un programa que se llama Postman para poder gestionar nuestras peticiones y listar las rutas.

**Postman**

Para conocer Postman te invito a:

Entrar a postman.com

Dentro, descargaremos el programa en la sección de "Download the desktop app".

Una vez listo, estaremos trabajando en nuevo "Request". Cada vez que hagamos una petición, debemos colocar en la sección de "METHOD", el tipo de método que necesitamos (GET, POST, PUT, DELETE) y luego la URL a la cual le estamos apuntando.

Obtendremos en la parte inferior la respuesta del servidor.

Avancemos con nuestro CRUD y desarrollemos el proceso.

**ExpressJS**

A partir de este momento, vamos a empezar a trabajar con **"Backend"**, en nuestro servidor, que tendrá comunicación con el cliente y con las bases de datos.

Usaremos ExpressJS.

**Estableciendo nuestro ambiente**

Para comenzar, generaremos un servidor con Node.js.

Comencemos estableciendo el siguiente código como base.

Generamos una carpeta que se llame ucamp-express.

Nos situamos en la carpeta y comenzamos estableciendo un archivo llamado index.js

Escribimos este comando:

npm init -y

Posteriormente, necesitaremos instalar estas dependencias:

express. Framework que nos permitirá gestionar nuestro ruteo y permitir el desarrollo del servidor, entre otras características.

body-parser. Librería que nos permitirá interpretar la petición que viene del cliente.

Para esto, ejecutaremos:

npm install express

npm install body-parser

También, instalaremos, de manera global nodemon. Este nos permitirá reiniciar automáticamente nuestro servidor a partir de cambios en nuestro código.

npm install -g nodemon

Nos situamos en package.json y reemplazos en la zona de scripts este código:

...

"scripts": {

"start": "nodemon index.js"

},

...

Con esto, cuando ejecutemos npm run start ejecutaremos el archivo index.js a través de la librería global nodemon.

Nos vamos a index.js

Dividiremos nuestra archivo en una estructura de 4 secciones y en cada una agregaremos sentencias específicas.

// 1. IMPORTACIONES

// Se refiere a todas las dependencias del

// proyecto que se utilizarán en este archivo.

const express = require('express')

const app = express()

const bodyParser = require("body-parser")

// 2. MIDDLEWARES

// Se refiere a todas las funciones

// que se ejecutarán antes de tocar las rutas.

app.use(bodyParser.urlencoded({extended:true}))

// 3. RUTEO

// Las rutas establecidas para nuestro servidor

// LEER TODAS LAS PERSONAS

app.get("/", (req, res) => {

const data = [{

id: 0,

nombre: "Mike",

pais: "México"

}]

res.json(data)

})

// CREAR UNA PERSONA

app.post("/", (req, res) => {

const { nombre, pais } = req.body

const data = [{

id: 0,

nombre: "Mike",

pais: "México"

}]

data.push({nombre, pais})

res.json(data)

})

// ACTUALIZAR UNA PERSONA

app.put("/", (req, res) => {

const { id, nombre, pais } = req.body

const data = [{

id: 0,

nombre: "Mike",

pais: "México"

}]

const dataFiltered = data.map((e) => {

return e.id === id ? {

id,

nombre,

pais

} : null

})

res.json(dataFiltered)

})

// BORRAR UNA PERSONA

app.delete("/", (req, res) => {

const {id} = req.body

const data = [{

id: 0,

nombre: "Mike",

pais: "México"

}]

const dataFiltered = data.filter((e) => {

return e.id !== id

})

res.json(dataFiltered)

})

// 4. SERVIDOR

app.listen(3000, () => {

console.log("El servidor está de pie")

})

**Leer elementos**

Cada una de las rutas ejecuta una respuesta con los datos que se solicitan en la petición.

Vamos a Postman y ejecutemos GET.

Observa que obtenemos de regreso nuestro arreglo que incluye un objeto con el id y sus datos.

Postman nos permitirá conectarnos a una ruta específica y devolverá una respuesta.

**Crear elemento**

Probemos con POST. Para este método, es importante que establezcamos en la sección de "BODY" dentro de Postman los datos de nuestra petición, las cuales simularán un formulario. Usaremos la opción "JSON" para generarlo.

**Editar elementos**

Ahora, editaremos un elemento. Para esto, enviaremos un objeto que identificará el ID dentro del backend y hará los cambios respectivos en nombre y país, enviados desde la petición.

**Borrar elementos**

Por último, ejecutaremos el borrado de un elemento.

Enviaremos únicamente el ID desde Postman, enfocándonos en el método DELETE, y borrará el registro que tenemos guardado en la ruta, generándolo como respuesta.

**¡Muy bien!, has llegado al final del contenido de esta semana, espero hayas aprendido. Recuerda que los coaches estan disponibles para aclarar tus dudas.**

**¡Nos vemos el sábado! 😃**