Introducción a Express Js

{desafío}
latam_

Activen las cámaras los que puedan y pasemos asistencia

- Objetivo de la sesión
- Conceptualización
- Ejercitación
- Reflexión

Inicio

{desafío}
latam_

Activación de conceptos

Contesta la pregunta correctamente y gana un punto

Instrucciones:

- Se realizará una pregunta, el primero en escribir “YO” por el chat, dará su respuesta al resto de la clase.
- El docente validará la respuesta.
- En caso de que no sea correcta, dará la oportunidad a la segunda persona que dijo “Yo”.
- Cada estudiante podrá participar un máximo de 2 veces.
- Al final, el/la docente indicará el 1º, 2º y 3º lugar.
- Esta actividad no es calificada, es solo una dinámica para recordar los conceptos clave para abordar esta sesión.

Activación de conceptos

```
const tareas = [
    { text: 'Ir al gimnasio' },
    { text: 'Buscar al niño al colegio' },
    { text: 'Pagar los gastos comunes' },
    { text: 'Cargar bencina' }
]

fs.writeFileSync('tareas.json', JSON.stringify(tareas) )
```

¿Qué hace JSON.stringify en el siguiente código?

Activación de conceptos

```
const fs = require('fs')

const tareas = fs.readFileSync('tareas.json', 'utf8')
tareas.forEach( (tarea) => {
    console.log(tarea)
})
```

¿Qué problema hay con este código?

{desafío} latam_

Activación de conceptos

```
const { saludar, darLasGracias } = require('./modales.js')
```

¿Cómo se deben exportar las siguientes funciones para poder importarlas de esta forma?

{desafío} latam_

Activación de conceptos

Primer lugar:
_____

Segundo lugar:
_____

Tercer lugar:
_____

{desafío} latam_

Objetivos

Objetivo general:

Levantar un servidor con Express Js que ofrezca una API REST para leer y agregar datos en un archivo JSON local

{desafío} latam_

Objetivos

/* Crear desde el backend una ruta GET usando Express Js */

/* Crear desde el backend una ruta GET para listar un conjunto de recursos en Express Js */

/* Crear una ruta POST en un backend para agregar datos en un JSON local */

Desarrollo

{desafío}
latam_

/* ¿Qué es Express Js? */

Introducción a Express Js

¿Qué es Express Js?

Express Js es un framework de JavaScript para la creación de servidores y API REST en un entorno Node Js.

Es utilizado por varias de las empresas más grandes del mundo como PayPal, Uber, Netflix, LinkedIn, IBM y muy probablemente también se esté usando en tu aplicación favorita.

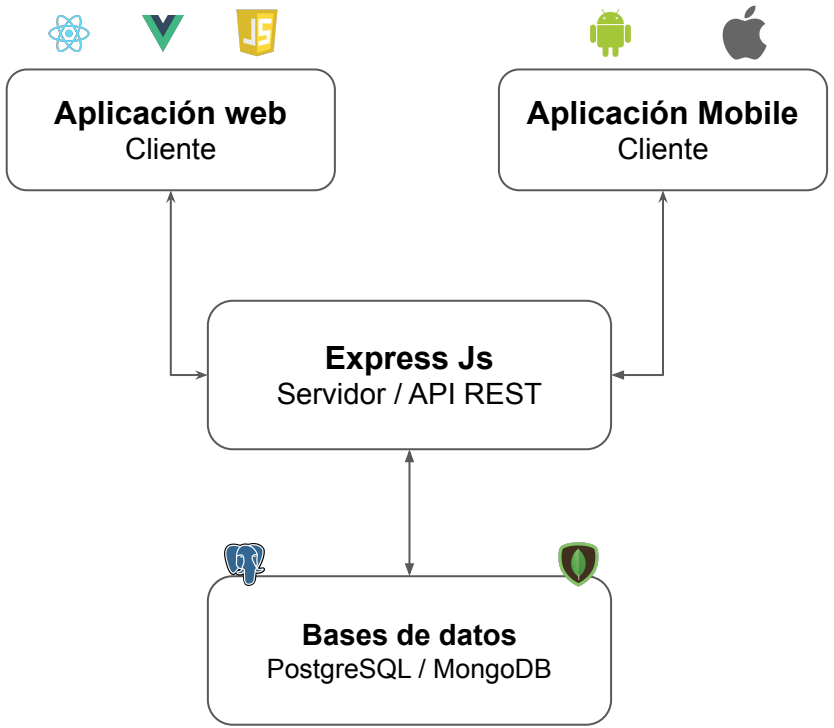
Los desarrolladores que prefieren Express Js como herramienta de trabajo típicamente lo ocupan por ser significativamente minimalista, con una sintaxis intuitiva y declarativa, además del poco tiempo de desarrollo y líneas de código que se necesita para empezar servidores y servicios web.

Introducción a Express Js

¿Qué son los servidores Web?

Un servidor web es una aplicación backend creada para recibir solicitudes de aplicaciones clientes y devolver información o inclusive sitios web.

Usualmente, desde el servidor utilizaremos bases de datos para la persistencia de datos, lo cual aprenderemos más adelante.



Introducción a Express Js

Instalación de Express Js

Para empezar a utilizar Express js primero hay que instalarlo por npm, para esto crea una nueva carpeta, dentro de la carpeta ejecuta la siguiente línea de comando para iniciar un proyecto NPM:

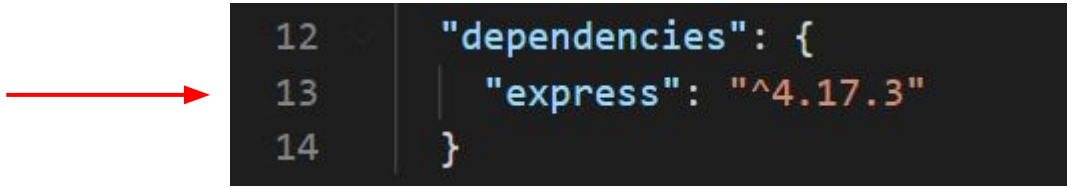
```
npm init -y
```

Asegúrate que el nombre de la carpeta no incluya mayúsculas

Ahora instalemos Express con la siguiente línea de comando:

```
npm install express
```

Luego de instalarse deberás ver "express" en el archivo *package.json*



Introducción a Express Js

Creación de rutas

Creemos nuestro primer proyecto titulado ***Hola mundo*** en donde creemos un servidor para devolver un string que diga “Hello World Express Js” en una ruta **GET /home**

Para esto crea un archivo *index.js* y escribe en él el siguiente código:

```
const express = require('express')
const app = express()

app.listen(3000, console.log("¡Servidor encendido!"))

app.get("/home", (req, res) => {
    res.send("Hello World Express Js")
})
```

Introducción a Express Js

Levantando mi servidor

Lo siguiente será ejecutar el archivo *index.js* por la terminal para levantar el servidor

```
node index.js
```

En donde deberás recibir el mensaje: **¡Servidor encendido!**

```
Brian@LAPTOP-7I8PV1OA MINGW64
$ node index.js
¡Servidor encendido!
```

Ahora que nuestro servidor está levantado, solo deberás abrir el navegador y escribir la siguiente URL:

http://localhost:3000/home

Y verás lo siguiente:

Introducción a Express Js

Explicando el código

Se importa el paquete **express** y se ejecuta para obtener un enrutador (app)

```
const express = require('express')
const app = express()
```

```
app.listen(3000, console.log("¡Servidor encendido!"))
```

Se especifica en qué puerto se levantará el servidor y se declara un mensaje por consola al levantarse

```
app.get("/home", (req, res) => {
    res.send("Hello World Express Js")
})
```

Se crea una ruta **GET** con un path **/home**

Que al consultarse devolverá un String con el texto "**Hello World Express Js**"

Introducción a Express Js

El enrutador de Express Js

Con la instancia ***app*** podremos crear todas las rutas que necesitemos. Solo debemos cumplir la siguiente sintaxis:

```
app.method( path, callback )
```

En donde:

- **app:** Es una instancia de Express.
- **method:** Es un método de solicitud HTTP en minúscula.
- **path:** Es la ruta que será consultada por la aplicación cliente, por ejemplo **/home**
- **callback:** Es la función que se ejecutará cuando la ruta sea consultada.

Introducción a Express Js

Apagar el servidor

Cuando levantamos un servidor de Express, este queda a la espera de solicitudes bloqueando la terminal:

```
Brian@LAPTOP-7I8PV1OA MINGW64 ~/Desktop/Clase de Express
$ node index.js
¡Servidor encendido!
```

Para que los cambios que hagamos en el código del servidor sean aplicados es necesario apagar el servidor y volverlo a levantar.

Para bajar o apagar el servidor, presiona **Crtl + C** en Windows y linux, o **Cmd + C** en Mac

```
Brian@LAPTOP-7I8PV1OA MINGW64 ~/Desktop/Clase de Express
$ node index.js
¡Servidor encendido!

Brian@LAPTOP-7I8PV1OA MINGW64 ~/Desktop/Clase de Express
$
```

{desafío} latam_

Introducción a Express Js

Actualizando cambios

Los cambios que realicemos en el código no se reflejarán hasta que reiniciemos el servidor. De forma alternativa podemos instalar el paquete nodemon para poder reflejar los cambios en el server sin tener que cerrar y abrirlo a cada rato.

Ejercicio

Crea una nueva carpeta titulada ***express a la obra*** y:

1. Inicia un proyecto de npm
2. Instala el paquete de express
3. Importa el paquete de express
4. Crear un servidor en el puerto 3000
5. Crea una ruta **GET /perfil** que devuelva un string con tu nombre
6. Consulta la ruta creada desde el navegador

Ejercicio
¡Manos al teclado!

{desafío} latam_

Objetivos

/* Crear una ruta GET en Express Js */ ✅

/* Devolver un JSON local en una ruta GET */

/* Recibir un payload en una ruta POST para agregar datos en un JSON local */

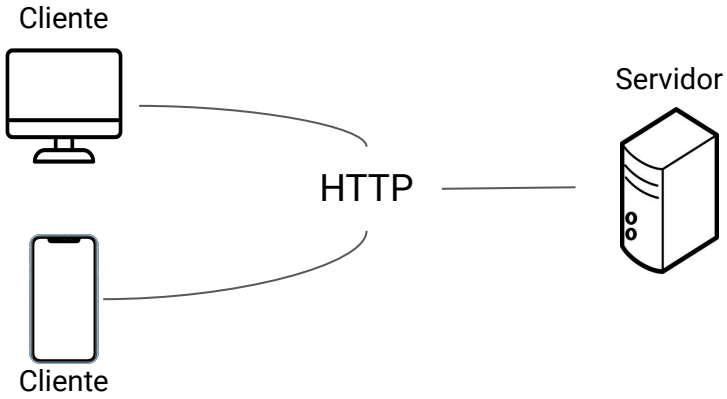
{desafío} latam_

Introducción a Express Js

Arquitectura Cliente - Servidor

La creación de servidores consiste en permitir la comunicación entre 2 aplicaciones que requieren compartir información o la posibilidad de ejecutar funciones a partir de consultas HTTP.

Esto mismo se conoce como la arquitectura **Cliente - Servidor**

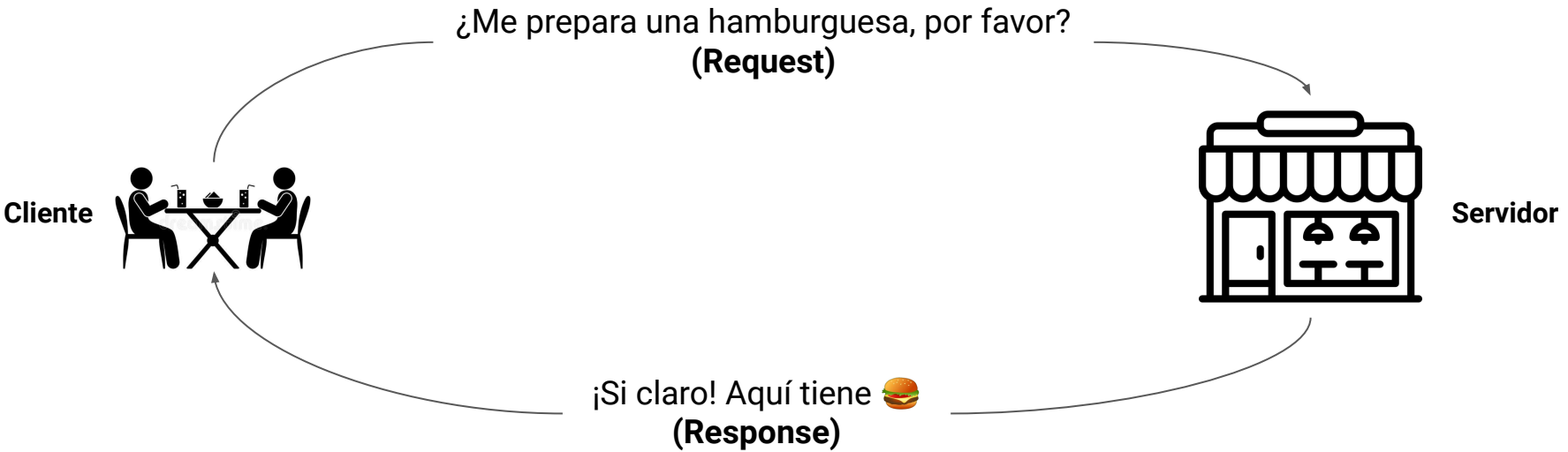


En donde tenemos un cliente que realiza una **consulta** y un servidor que procesa la consulta y emite una **respuesta.**

Introducción a Express Js

Arquitectura Cliente - Servidor

Pudiéramos comparar esta arquitectura con el proceso de pedidos en un restaurante:

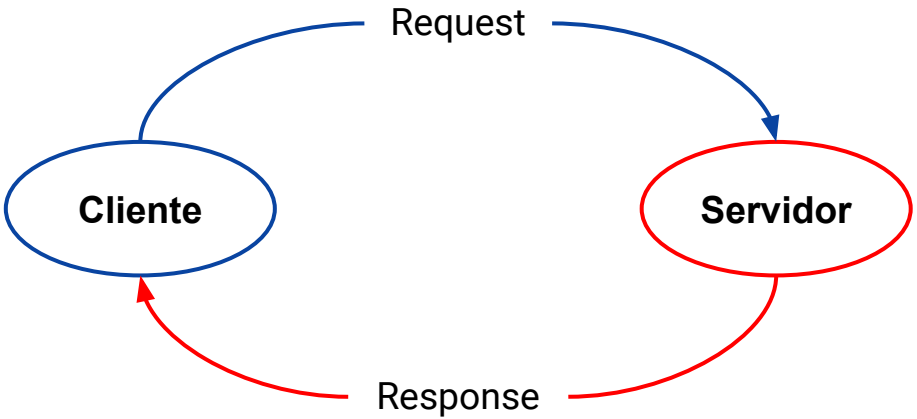


Introducción a Express Js

El objeto request(req) y response(res)

Los servidores disponibilizan datos y funciones a través de rutas y en cada una de ellas tenemos acceso a 2 parámetros conocidos como el **request** y el **response**.

En donde el **request** corresponde a la información de la consulta recibida en el servidor, mientras que el **response** será la respuesta que le daremos a las aplicaciones clientes.



Introducción a Express Js

El objeto request(req) y response(res)

En Express el request y el response son objetos que podemos utilizar para obtener el detalle de la consulta y poder dar respuesta:

Existen varios atributos y métodos en cada objeto, pero para esta clase solo ocuparemos los siguientes:

Request:

- **req.body:** Devuelve el payload o cuerpo de la consulta

Response:

- **res.send():** Devuelve como respuesta de la consulta lo que sea asignado como argumento
- **res.json():** Devuelve en formato JSON lo que sea asignado como argumento

Introducción a Express Js

Devolviendo un JSON en una ruta

Frecuentemente, crearemos una API REST para entregar información a partir de una consulta bajo el método **GET.**

Realicemos este ejercicio creando un archivo *productos.json* en la misma carpeta donde creamos el proyecto "**Hola mundo**":

```
[
    {
        "id": 1,
        "nombre": "Audifonos",
        "precio": 18990
    },
    {
        "id": 2,
        "nombre": "Aro de Luz",
        "precio": 21990
    }
]
```

{desafío} latam_

Introducción a Express Js

Devolviendo un JSON en una ruta

Ahora creemos una ruta **GET /productos** en nuestro servidor y utilicemos el objeto *response* de la función callback para devolver el JSON creado.

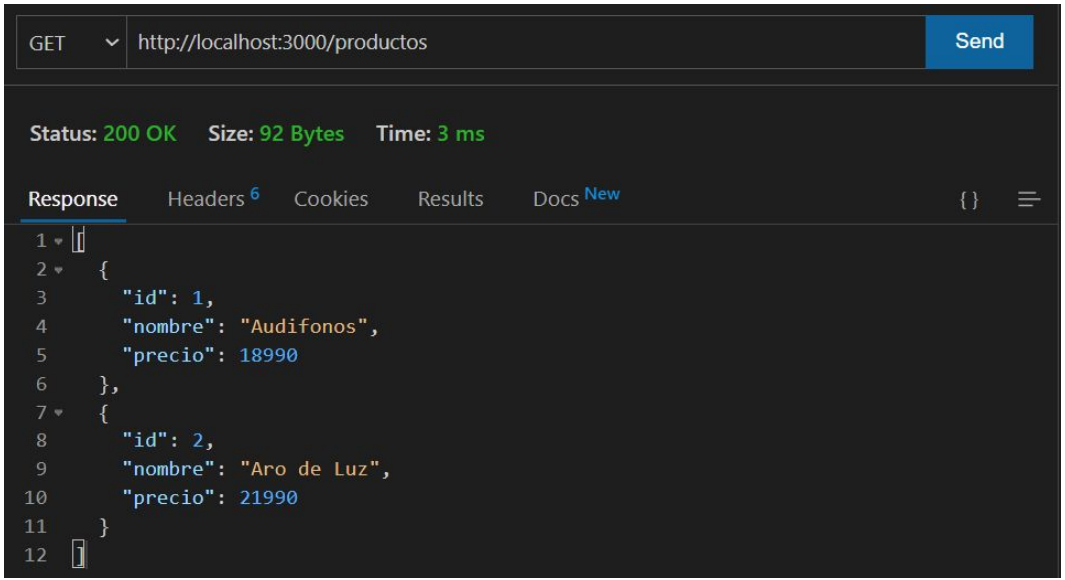
Para esto será necesario importar el módulo File System para leer el archivo *productos.json* y posteriormente devolverlo al cliente. El código del servidor quedaría así:

```
const express = require('express')
const app = express()
const fs = require('fs')

app.listen(3000, console.log("¡Servidor encendido!"))

app.get("/productos", (req, res) => {
    const productos = JSON.parse(fs.readFileSync("productos.json"))
    res.json(productos)
})
```

Introducción a Express Js

Devolviendo un JSON en una ruta

Ahora realicemos una consulta HTTP con la extensión Thunder Client de VSC consultando la siguiente ruta: http://localhost:3000/productos

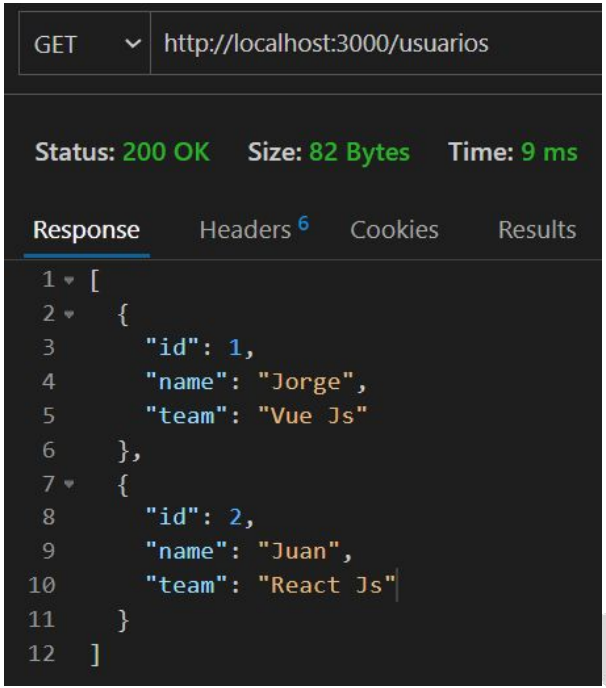


Si no tienes instalada la extensión, también puedes hacer la consulta de la ruta desde el navegador

Ejercicio

Continuando con el proyecto ***express a la obra***, crea una ruta **GET /usuarios** que devuelva el siguiente JSON:

```
[
  {
    "id": 1,
    "name": "Jorge",
    "team": "Vue Js"
  },
  {
    "id": 2,
    "name": "Juan",
    "team": "React Js"
  }
]
```



Ejercicio
¡Manos al teclado!

{desafío} latam_

Objetivos

/* Crear una ruta GET en Express Js */ ✅

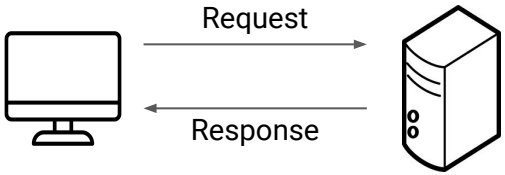
/* Devolver un JSON local en una ruta GET */ ✅

/* Recibir un payload en una ruta POST para agregar datos en un JSON local */

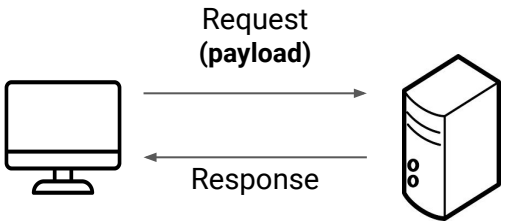
Introducción a Express Js

Recibiendo un payload

Hasta ahora solo hemos devuelto información a un cliente que consulta una de nuestras rutas:



Sin embargo, en muchas ocasiones necesitaremos recibir datos de parte del cliente:



El **payload** se entiende como la "carga" dentro de una consulta. En express esta carga se encuentra en el atributo ***body*** del objeto **request**

```
Request: {
  body: {
    "id": 3,
    "nombre": "Monitor",
    "precio": 59990
  }
}
```

Al recibir datos de parte del cliente podemos utilizarlos para por ejemplo registrar un nuevo producto en nuestra base de datos (por ahora solo un json local).

{desafío} latam_

Introducción a Express Js

Recibiendo un payload

Para poder leer el payload es necesario realizar una pequeña configuración en nuestra instancia app. Esta configuración consiste en ejecutar una función antes de cada ruta para parsear el contenido enviado desde el cliente. Para entender bien lo que vamos a hacer tenemos que conocer el concepto de middleware.

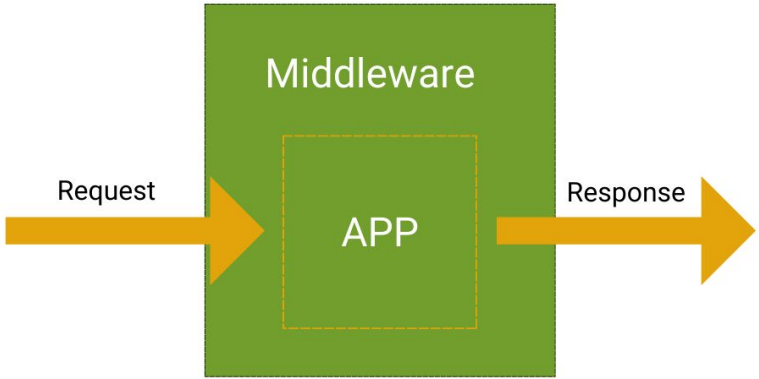
Introducción a Express Js

¿Qué son los middlewares?

Un middleware, particularmente un HTTP middleware, es una capa intermedia que envuelve la aplicación con la finalidad de procesar el request y el response.

Los middleware pueden ser usados para procesar fácilmente los payloads, añadir headers, comprimir los response, manejar autenticación.

Ahora los ocuparemos para leer el payload



Introducción a Express Js

Usando un middleware en Express

El siguiente diagrama nos ayudará a entender cómo funciona

Introducción a Express Js

¿Qué son los middlewares?

En Express Js los middlewares se declaran con el método .***use()*** *de express*, pasando como argumento el middleware.

Para eso agregamos la siguiente línea en nuestro servidor

```
const express = require('express')
const app = express()
const fs = require('fs')
app.listen(3000, console.log("¡Servidor encendido!))

app.use(express.json())

app.get("/productos", (req, res) => {
    const productos = JSON.parse(fs.readFileSync("productos.json"))
    res.json(productos)
})
```

Introducción a Express Js

Creación de una ruta POST

Ahora creemos una ruta **POST** que sea utilizada para agregar un nuevo producto en nuestro JSON local.

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

    const producto = req.body

    const productos = JSON.parse(fs.readFileSync("productos.json"))

    productos.push(producto)

    fs.writeFileSync("productos.json", JSON.stringify(productos))

    res.send("Producto agregado con éxito!")

})
```

Accedemos al payload de la consulta por medio del atributo body del request

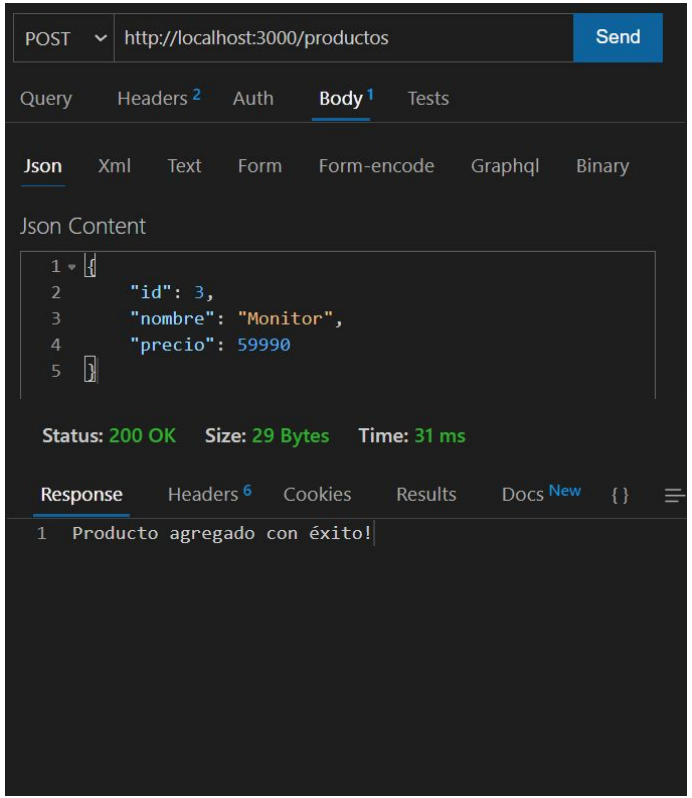
Sobreescribimos el archivo productos.json luego de agregar el nuevo producto

Introducción a Express Js

Agregando un nuevo producto

Probemos nuestra ruta **POST /productos** realizando una consulta HTTP con la extensión Thunder Client pasando como payload lo siguiente:

```
{
    "id": 3,
    "nombre": "Monitor",
    "precio": 59990
}
```



{desafío} latam_

Introducción a Express Js

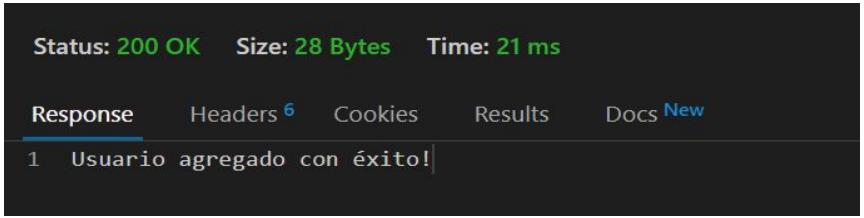
Agregando un nuevo producto

Ahora si revisamos nuestro archivo local *productos.json* notaremos que el Monitor fue agregado.

Ejercicio

Continuando con el proyecto ***express a la obra***, crea una ruta **POST /usuarios** que reciba el registro de un nuevo usuario y lo agregue al archivo *usuarios.json*.

Deberás probar esta ruta con la extensión Thunder Client



{desafío} latam_

Ejercicio
¡Manos al teclado!

Objetivos

/* Crear una ruta GET en Express Js */ ✅

/* Devolver un JSON local en una ruta GET */ ✅

/* Recibir un payload en una ruta POST para agregar datos en un JSON local */ ✅

Cierre

{desafío}
latam_

Reflexionemos

¿Existe algún concepto que no hayas comprendido?

{desafío} latam_

¿Qué sigue?

- Revisar la guía que trabajarán de forma autónoma.
- Revisar en conjunto el desafío.

{desafío} latam_

{desafío} latam_

Academia de talentos digitales