



**¡Les damos la
bienvenida!**

¿Comenzamos?

Unidad 04. PROGRAMACIÓN BACKEND

Desarrollo de backend Avanzado

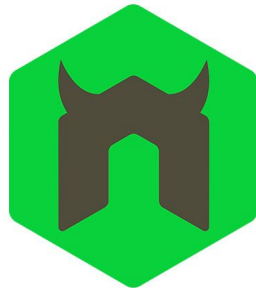
Objetivos de la clase

- **Entender** el protocolo HTTP
- **Aprender** a crear un servidor utilizando el módulo nativo http
- **Crear** un servidor de express
- **Profundizar** sobre las peticiones GET de un servidor express.

Objetivos de la clase

- **Conocer** los status del protocolo HTTP
- **Comprender** el concepto de API REST
- **Conocer** los métodos POST, PUT, DELETE y utilizarlos con POSTMAN o ThunderClient
- **Profundizar** de manera práctica sobre los métodos POST, PUT, DELETE

Instalar nodemon de manera global



Como nuestro servidor se mantiene escuchando todo el tiempo, los cambios que hagamos en el código no se reflejan automáticamente. Por ello **tenemos que apagar el servidor y levantarlo nuevamente.**

Sólo así podemos visualizar los cambios que hagamos en el código (así sea un simple `console.log()`)

Nodemon nos permitirá **reiniciar automáticamente** el servidor en cuanto detecta que hay cambios en el código. De esta manera, podemos concentrarnos en el código, sin tener que realizar el reinicio manual cada vez que queremos ver algo.

Para instalarlo, basta correr el comando:

```
npm install -g nodemon
```



Ejemplo en vivo

- ✓ Crear un servidor con el módulo nativo de nodejs "http". Setear una respuesta que contenga el mensaje "¡Mi primer hola mundo desde backend!"
- ✓ El servidor debe escuchar en el puerto 8080. (Correr con nodemon)
- ✓ Probar el servidor desde el navegador.
- ✓ Hacer algún cambio en código y corroborar que se reinicie automáticamente.

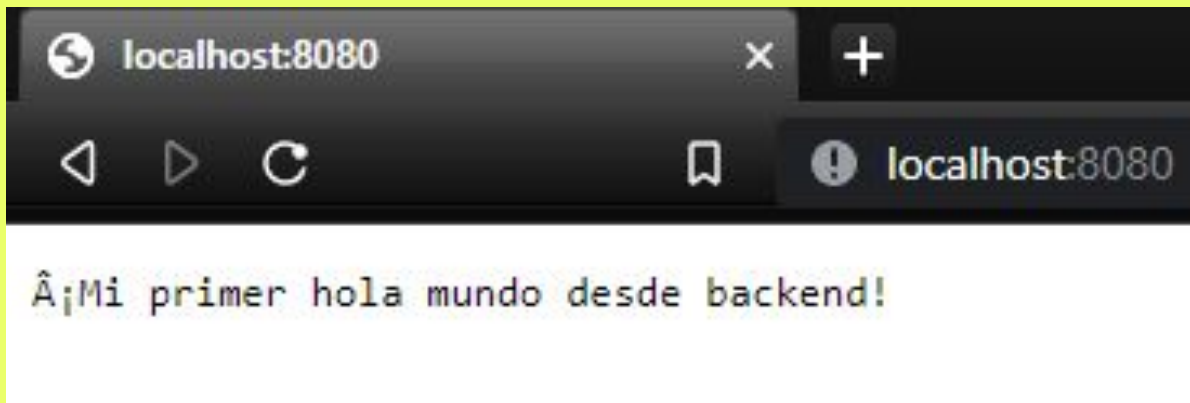
Parte 1: Realizando el servidor

```
const http = require('http');

const server = http.createServer((request, response) => {
  response.end("¡Mi primer hola mundo desde backend!");
})

server.listen(8080, () => {
  console.log("Listening on port 8080")
})
```

Parte 2: Probando desde navegador

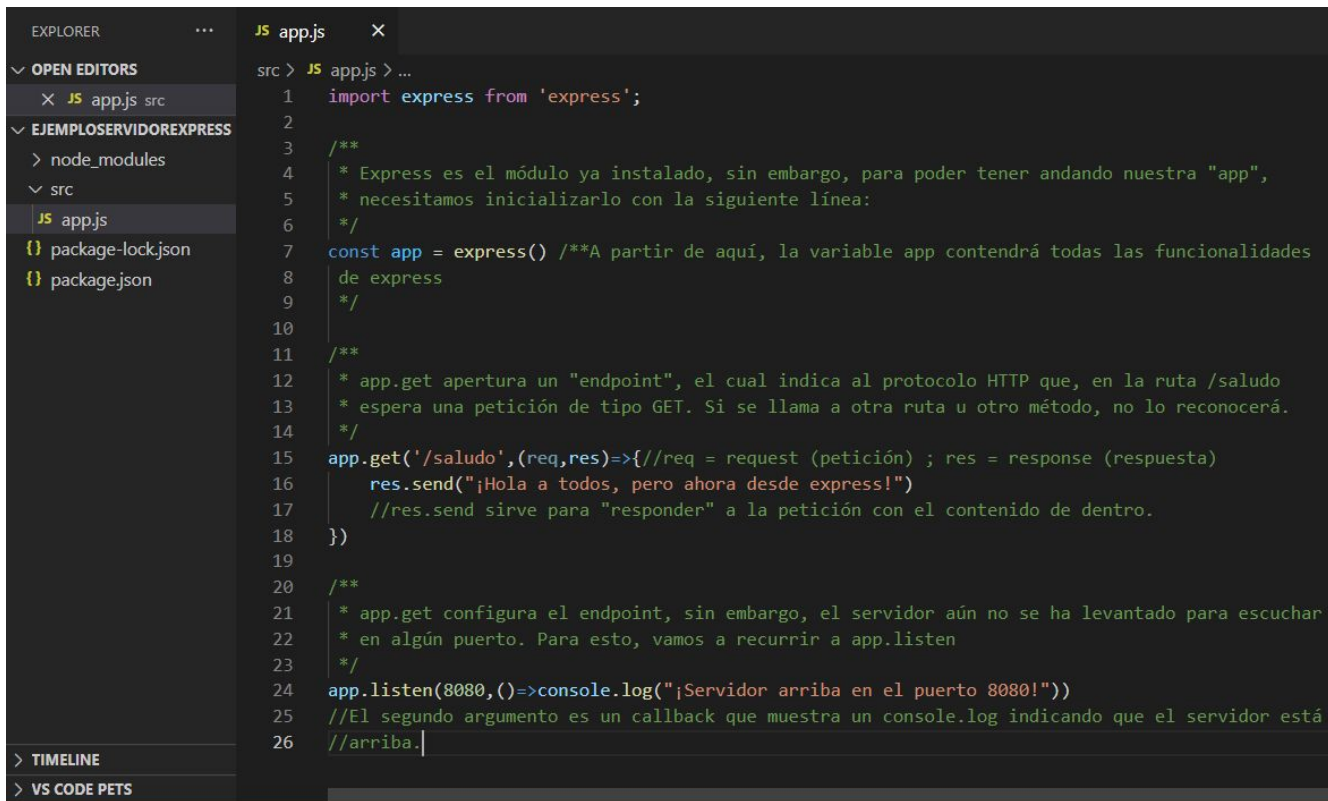




Ejemplo de una consulta en Express

- ✓ Estructurar un servidor basado en express, el cual escuche peticiones en el puerto 8080
- ✓ Realizar una función para el método GET en la ruta '/saludo', el cual responderá con "¡Hola a todos, pero ahora desde express!"
- ✓ Ejecutar con nodemon y probar en el navegador el endpoint generado

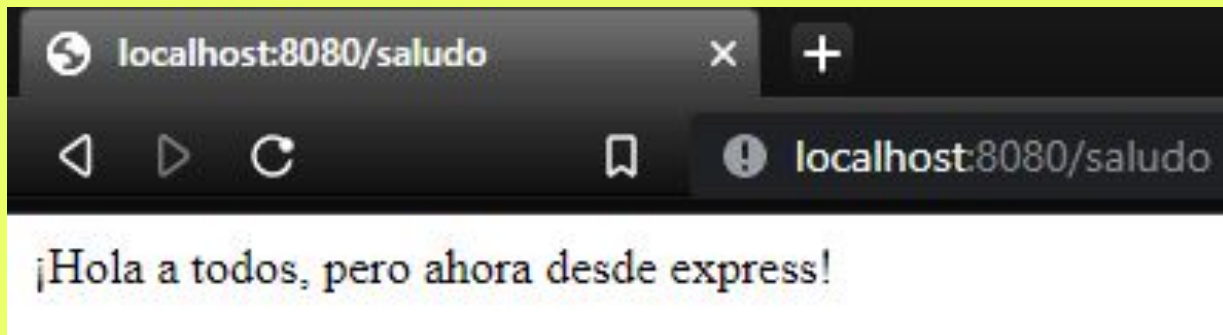
Parte 1: implementando el proyecto



The image shows a VS Code editor window with a dark theme. On the left, the Explorer sidebar shows a file tree with 'src' containing 'app.js', 'package-lock.json', and 'package.json'. The main editor area shows the content of 'app.js' with line numbers 1 through 26. The code imports 'express' and sets up a basic server with a GET endpoint at '/saludo' that responds with '¡Hola a todos, pero ahora desde express!'. The server listens on port 8080.

```
src > JS app.js > ...
1  import express from 'express';
2
3  /**
4   * Express es el módulo ya instalado, sin embargo, para poder tener andando nuestra "app",
5   * necesitamos inicializarlo con la siguiente línea:
6   */
7  const app = express() /**A partir de aquí, la variable app contendrá todas las funcionalidades
8   de express
9   */
10
11 /**
12  * app.get apertura un "endpoint", el cual indica al protocolo HTTP que, en la ruta /saludo
13  * espera una petición de tipo GET. Si se llama a otra ruta u otro método, no lo reconocerá.
14  */
15  app.get('/saludo',(req,res)=>{/req = request (petición) ; res = response (respuesta)
16    res.send("¡Hola a todos, pero ahora desde express!")
17    //res.send sirve para "responder" a la petición con el contenido de dentro.
18  })
19
20 /**
21  * app.get configura el endpoint, sin embargo, el servidor aún no se ha levantado para escuchar
22  * en algún puerto. Para esto, vamos a recurrir a app.listen
23  */
24  app.listen(8080,()=>console.log("¡Servidor arriba en el puerto 8080!"))
25  //El segundo argumento es un callback que muestra un console.log indicando que el servidor está
26  //arriba.
```

Parte 2: Probando en el navegador en endpoint especificado





Otras respuestas con express

Conociendo otros tipos de elementos que podemos responder con express.

Duración: 10 min



ACTIVIDAD EN CLASE

Otras respuestas con express

Crear un proyecto basado en express js, el cual cuente con un servidor que escuche en el puerto 8080. Además de configurar los siguientes endpoints:

- ✓ El endpoint del método GET a la ruta `/bienvenida` deberá devolver un html con letras en color azul, en una string, dando la bienvenida.
- ✓ El endpoint del método GET a la ruta `/usuario` deberá devolver un objeto con los datos de un usuario falso: `{nombre, apellido, edad, correo}`





Ejemplo en vivo: Caso práctico de uso de params

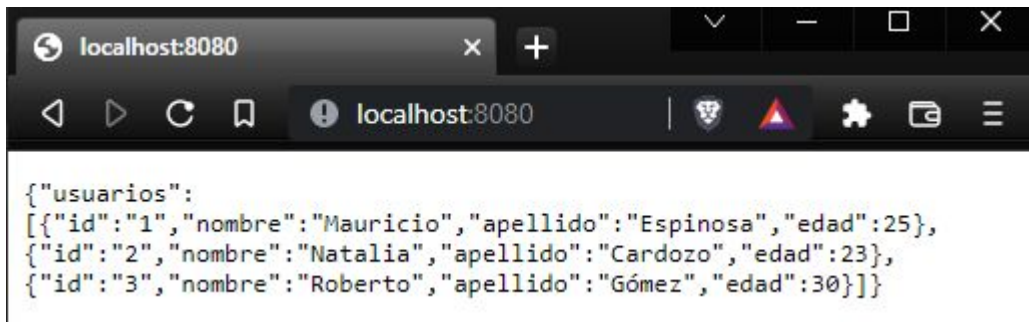
- ✓ Dado un arreglo de objetos de tipo usuario, realizar un servidor en express que permita obtener dichos usuarios.
- ✓ La ruta raíz '/' debe devolver todos los usuarios
- ✓ la ruta /:userId debe devolver sólo al usuario con dicho Id.

Ejemplo: Implementación

JS casoPrácticoParams.js X

```
src > JS casoPrácticoParams.js > [0] usuarios
1  import express from 'express';
2
3  const app = express();
4  const usuarios = [
5    {id:"1",nombre:"Mauricio",apellido:"Espinosa",edad:25},
6    {id:"2",nombre:"Natalia",apellido:"Cardozo",edad:23},
7    {id:"3",nombre:"Roberto",apellido:"Gómez",edad:30}
8  ]
9
10 app.get('/',(req,res)=>{
11   res.send({usuarios}) //Se recomienda enormemente mandar los datos en formato objeto en lugar
12   //de enviarlos como un array solo, esto permite que, si vamos a meter más info en el futuro
13   //no tengamos que cambiar el tipo de respuesta de lado del cliente.
14 })
15 app.get('/:idUserio',(req,res)=>{
16   let idUsuario = req.params.idUsuario; //Obtenemos el id del usuario a trabajar
17   /**
18    * Procedemos a buscar al usuario que tenga el id pasado por params.
19    */
20   let usuario = usuarios.find(u=>u.id===idUserio);
21   //Si no encuentra al usuario, debe finalizar la función devolviendo un error
22   if(!usuario) return res.send({error:"Usuario no encontrado"})
23
24   //En caso de que no haya finalizado la función, significa que el usuario sí se encontró.
25   res.send({usuario})
26 })
27 app.listen(8080,()=>console.log("Listo para probar caso práctico"))
```


Prueba ruta raíz: Debe devolver todos los usuarios.



A screenshot of a web browser window with the address bar showing 'localhost:8080'. The browser displays a JSON response from the root endpoint. The response is a JSON object with a key 'usuarios' containing an array of three user objects. Each user object has keys for 'id', 'nombre', 'apellido', and 'edad'.

```
{
  "usuarios": [
    {
      "id": "1",
      "nombre": "Mauricio",
      "apellido": "Espinosa",
      "edad": 25
    },
    {
      "id": "2",
      "nombre": "Natalia",
      "apellido": "Cardozo",
      "edad": 23
    },
    {
      "id": "3",
      "nombre": "Roberto",
      "apellido": "Gómez",
      "edad": 30
    }
  ]
}
```

Prueba con parámetro válido: Debe encontrar al usuario y devolver sólo ese usuario.



A screenshot of a web browser window with the address bar showing 'localhost:8080/1'. The browser displays a JSON response for the user with ID '1'. The response is a JSON object with a key 'usuario' containing a single user object.

```
{
  "usuario": {
    "id": "1",
    "nombre": "Mauricio",
    "apellido": "Espinosa",
    "edad": 25
  }
}
```

Prueba con parámetro inválido: Debe arrojar error al no encontrar usuario con ese id.



A screenshot of a web browser window with the address bar showing 'localhost:8080/123232323'. The browser displays a JSON response indicating that the user was not found. The response is a JSON object with a key 'error' containing the string 'Usuario no encontrado'.

```
{
  "error": "Usuario no encontrado"
}
```



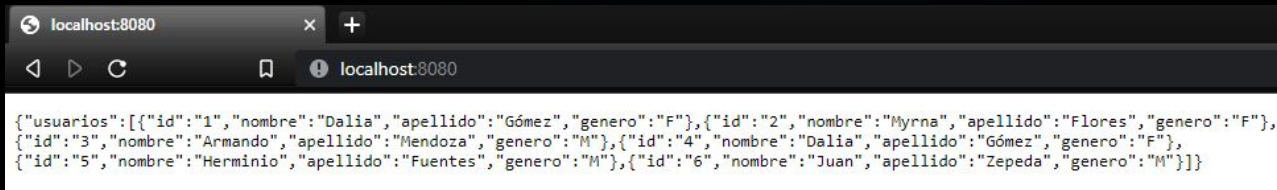

Ejemplo en vivo: Caso práctico de uso de req.query

- ✓ Dado un arreglo de objetos de tipo usuario, vamos a hacer un filtro por género
- ✓ La ruta raíz '/' debe devolver todos los usuarios, pero ahora colocaremos un query param con ?, indicando que queremos un género específico. En caso de enviarlo sin query, debe devolver a todos los usuarios.

Ejemplo:
implementando
endpoint que
utiliza
req.query

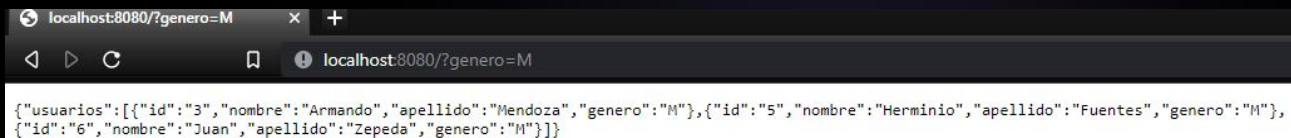
```
JS casoPrácticoQuery.js X
src > JS casoPrácticoQuery.js > app.get('/') callback
1  import express from 'express';
2
3  const app = express();
4
5  app.use(express.urlencoded({extended:true}))
6
7  //Filtraremos por género masculino (M) y femenino (F)
8  const usuarios = [
9    {id:"1",nombre:"Dalia",apellido:"Gómez",genero:"F"},
10   {id:"2",nombre:"Myrna",apellido:"Flores",genero:"F"},
11   {id:"3",nombre:"Armando",apellido:"Mendoza",genero:"M"},
12   {id:"4",nombre:"Dalia",apellido:"Gómez",genero:"F"},
13   {id:"5",nombre:"Herminio",apellido:"Fuentes",genero:"M"},
14   {id:"6",nombre:"Juan",apellido:"Zepeda",genero:"M"},
15 ]
16
17
18 app.get('/',(req,res)=>{
19   let genero = req.query.genero;
20   //Si no se ingresó género, o el género no es M ni es F, no vale el filtro.
21   if(!genero||(genero!="M"&&genero!="F")) return res.send({usuarios})
22   //En caso contrario, continuamos con el proceso de filtro.
23   let usuariosFiltrados = usuarios.filter(usuario=>usuario.genero===genero);
24   res.send({usuarios:usuariosFiltrados})
25 })
26
27 app.listen(8080,()=>console.log("Preparado para hacer filtros"))
```

Llamar al endpoint sin query permite que lleguen todos los usuarios sin filtrar



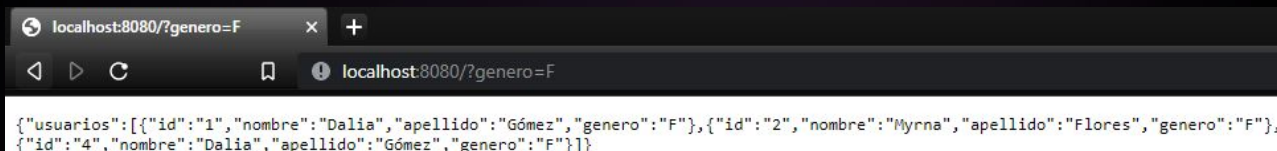
```
{
  "usuarios": [
    {
      "id": "1",
      "nombre": "Dalia",
      "apellido": "Gómez",
      "genero": "F"
    },
    {
      "id": "2",
      "nombre": "Myrna",
      "apellido": "Flores",
      "genero": "F"
    },
    {
      "id": "3",
      "nombre": "Armando",
      "apellido": "Mendoza",
      "genero": "M"
    },
    {
      "id": "4",
      "nombre": "Dalia",
      "apellido": "Gómez",
      "genero": "F"
    },
    {
      "id": "5",
      "nombre": "Herminio",
      "apellido": "Fuentes",
      "genero": "M"
    },
    {
      "id": "6",
      "nombre": "Juan",
      "apellido": "Zepeda",
      "genero": "M"
    }
  ]
}
```

Enviando el query con genero = M, notamos cómo devuelve sólo los usuarios con género M.



```
{
  "usuarios": [
    {
      "id": "3",
      "nombre": "Armando",
      "apellido": "Mendoza",
      "genero": "M"
    },
    {
      "id": "5",
      "nombre": "Herminio",
      "apellido": "Fuentes",
      "genero": "M"
    },
    {
      "id": "6",
      "nombre": "Juan",
      "apellido": "Zepeda",
      "genero": "M"
    }
  ]
}
```

Mismo caso para el género F.



```
{
  "usuarios": [
    {
      "id": "1",
      "nombre": "Dalia",
      "apellido": "Gómez",
      "genero": "F"
    },
    {
      "id": "2",
      "nombre": "Myrna",
      "apellido": "Flores",
      "genero": "F"
    },
    {
      "id": "4",
      "nombre": "Dalia",
      "apellido": "Gómez",
      "genero": "F"
    }
  ]
}
```

¡Acabamos de hacer un filtro de usuarios perfectamente funcional!



Break

¡10 minutos y volvemos!



Ejemplo en vivo: Integrando todos los métodos

- ✓ Se agregará al código de explicación un método GET al mismo endpoint, con el fin de completar los 4 métodos principales.
- ✓ Se realizará un flujo completo con POSTMAN donde podremos ver trabajando a todos los endpoints en conjunto, revisando,



Servidor con GET, POST, PUT, DELETE

Servidor con integración de todos los métodos

Duración: 20 min

Se trabajará de forma individual en la sala general



ACTIVIDAD EN CLASE

Servidor con GET, POST, PUT, DELETE

Dada la frase: “Frase inicial”, realizar una aplicación que contenga un servidor en express, el cual cuente con los siguientes métodos:

- ✓ GET '/api/frase': devuelve un objeto que como campo 'frase' contenga la frase completa
- ✓ GET '/api/palabras/:pos': devuelve un objeto que como campo 'buscada' contenga la palabra hallada en la frase en la posición dada (considerar que la primera palabra es la #1).



ACTIVIDAD EN CLASE

- ✓ POST '/api/palabras': recibe un objeto con una palabra bajo el campo 'palabra' y la agrega al final de la frase. Devuelve un objeto que como campo 'agregada' contenga la palabra agregada, y en el campo 'pos' la posición en que se agregó dicha palabra.
- ✓ PUT '/api/palabras/:pos': recibe un objeto con una palabra bajo el campo 'palabra' y reemplaza en la frase aquella hallada en la posición dada. Devuelve un objeto que como campo 'actualizada' contenga la nueva palabra, y en el campo 'anterior' la anterior.
- ✓ DELETE '/api/palabras/:pos': elimina una palabra en la frase, según la posición dada (considerar que la primera palabra tiene posición #1).
- ✓ Utilizar POSTMAN para probar funcionalidad

Muchas gracias.

#DemocratizandoLaEducación