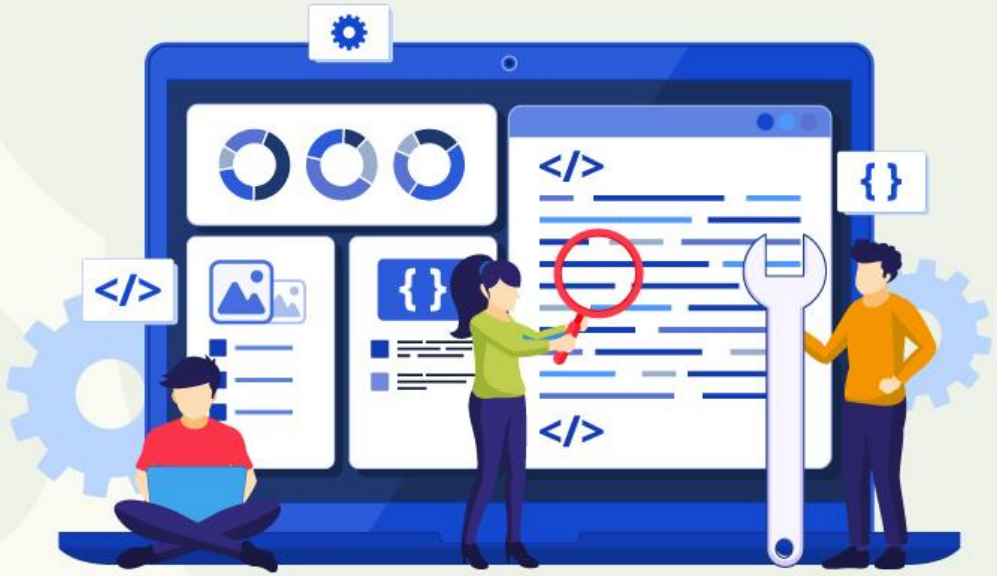
CICLO 4a

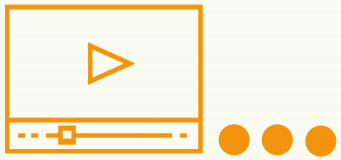
[FORMACIÓN POR CICLOS]

# Desarrollo de APLICACIONES WEB

## Tutorial de creación de un crud




UNIVERSIDAD DE ANTIOQUIA
Facultad de Ingeniería



Vamos a crear una aplicación sencilla que permitirá realizar un CRUD (crear, leer, actualizar y borrar) para libros. Inicialmente no conectaremos directamente con base de datos, lo que quiere decir que los datos guardados se perderán cuando se cierre o actualice la aplicación. Este CRUD estará orientado al uso de API REST usando Node y Express.

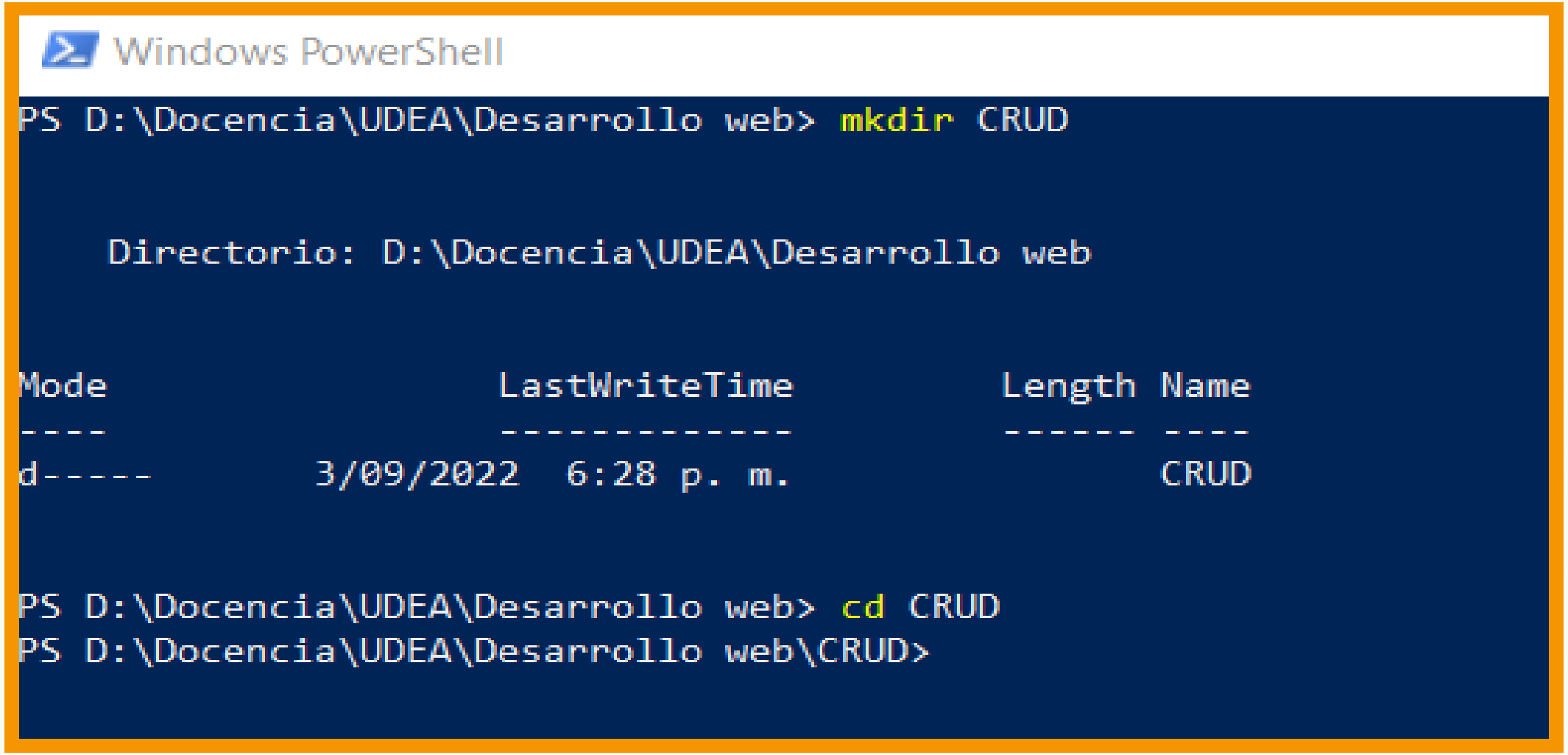
Para iniciar crearemos el proyecto y haremos una prueba de funcionamiento antes de entrar en materia.

Ingresa a powershell, tu terminal cmd, o la terminal de visual code.

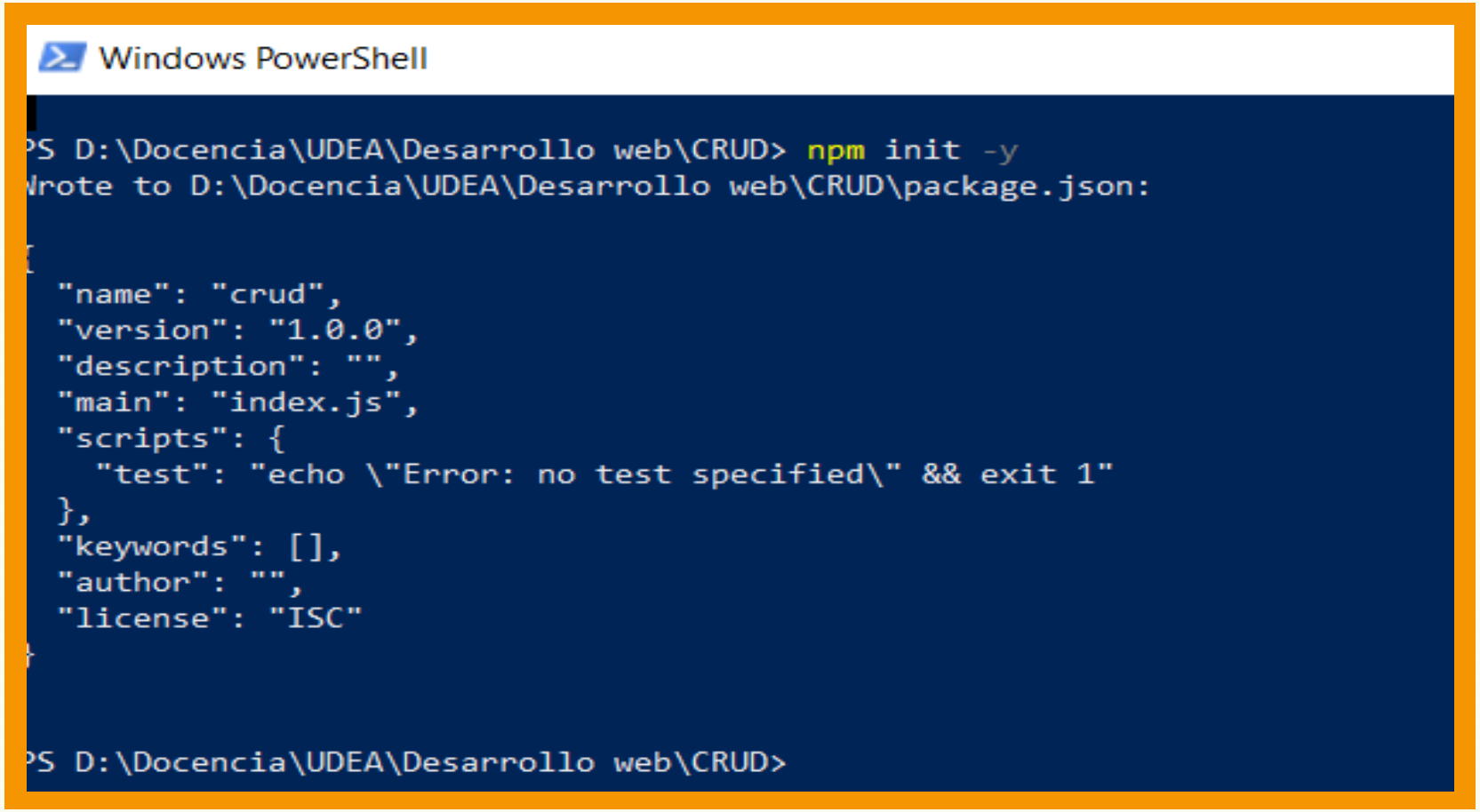
Crea la carpeta del proyecto e ingresa a ella.

Crea -> mkdir NombreCarpeta

Ingresa a carpeta -> cd NombreCarpeta


Windows PowerShell
PS D:\Docencia\UDEA\Desarrollo web> mkdir CRUD
Directorio: D:\Docencia\UDEA\Desarrollo web
Mode LastWriteTime Length Name
---- ------------- ------ ----
d----- 3/09/2022 6:28 p. m. CRUD
PS D:\Docencia\UDEA\Desarrollo web> cd CRUD
PS D:\Docencia\UDEA\Desarrollo web\CRUD>

Para que nuestro proyecto sea creado bajo node.js lo inicializamos con el comando NPM init -y. El final -y nos permite crear los datos del proyecto por defecto. Veremos que se crea un archivo .json con los metadatos más relevantes del proyecto. Este archivo es de solo consulta, no lo vamos a afectar.



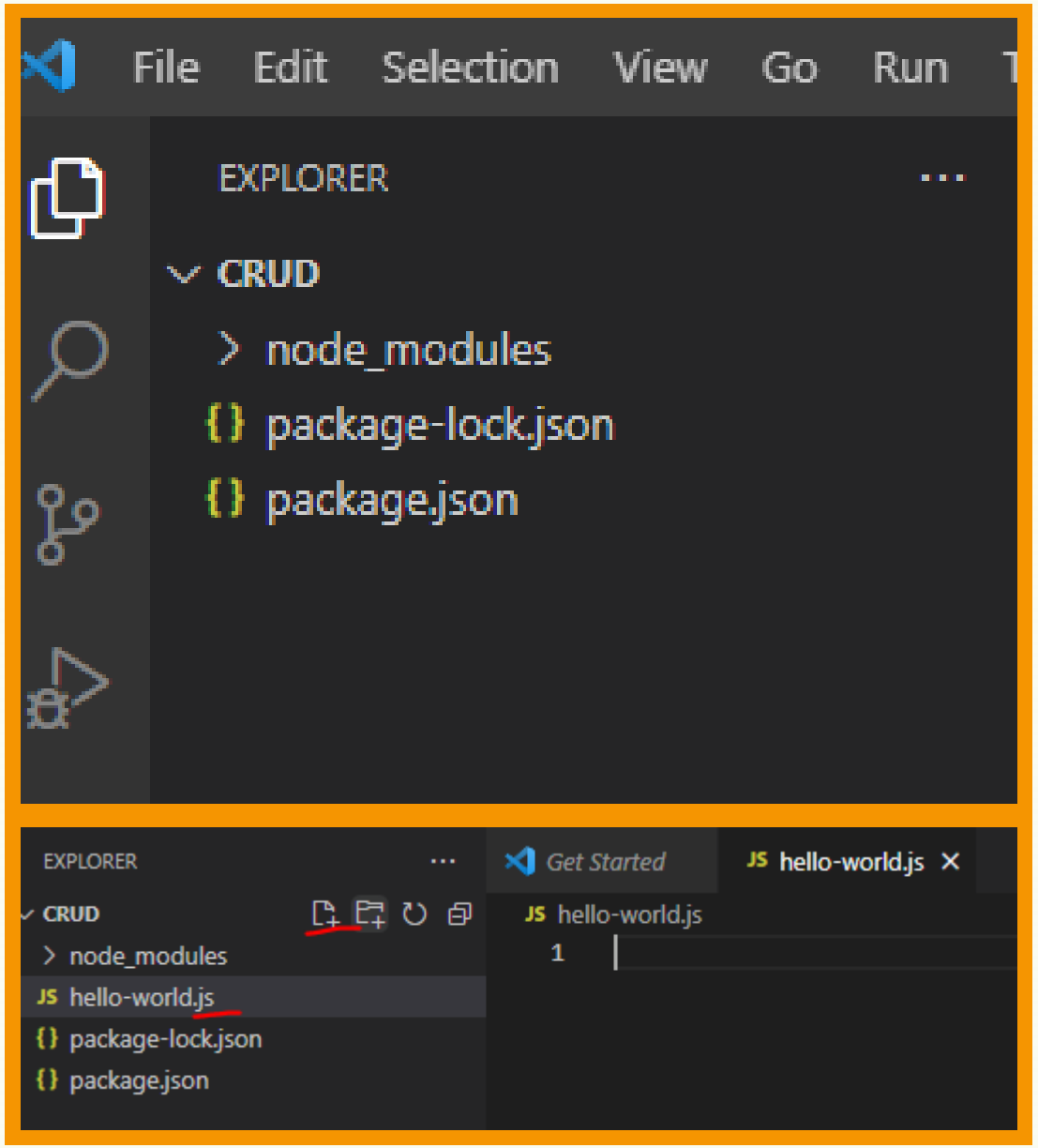
Ahora instalaremos Express con la línea npm install --save express:

```
PS D:\Docencia\UDEA\Desarrollo web\CRUD> npm install --save express

added 57 packages, and audited 58 packages in 6s

7 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
PS D:\Docencia\UDEA\Desarrollo web\CRUD>
```

3

Al realizar esto en nuestro proyecto se crea la carpeta node_modules. Si deseas subir tu proyecto a Git, esta es una carpeta que se recomienda que ignores.

En este punto ya podremos crear un end-point simple para visualizar el funcionamiento del proyecto. Iniciemos por crear un archivo hello-world.js y en él importaremos el marco Express:



En este archivo importamos el marco Express, creamos una instancia del mismo y configuramos el puerto en el que correrá el proyecto:

```
const express = require('express'); //importamos el marco de trabajo
const app = express(); //se crea la instancia de la aplicación Express
const port = 3001; // Se configura el puesto
```

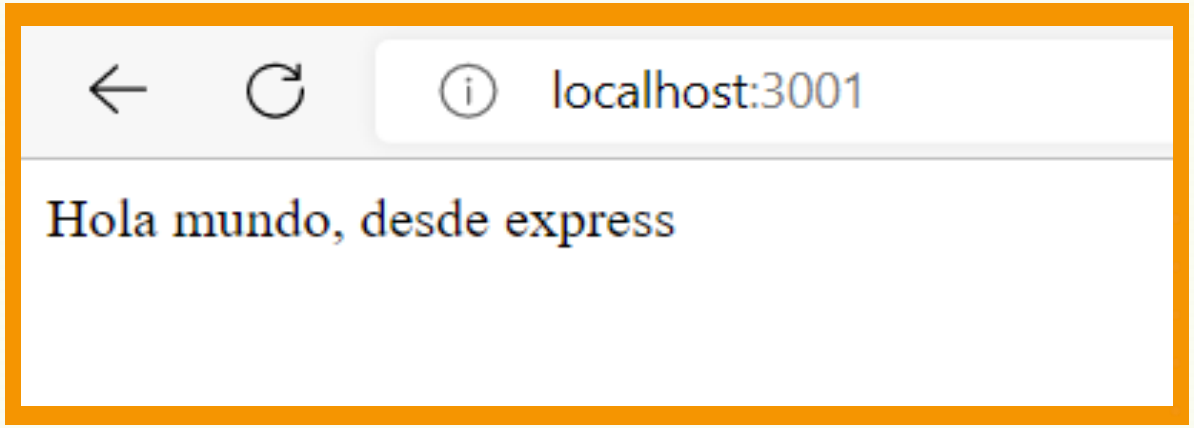
Ahora es posible crear un simple GET end-point, de tal manera que cuando un usuario llegue al end-point con una solicitud tipo GET, se retornará el mensaje “Hola mundo, desde Express”. Veremos este mensaje tanto en el navegador como en la consola.

```
app.get('/', (req, res) => {
    res.send('Hola mundo, desde Express');
});


app.listen(port, () => console.log(`La app hello world esta en el puerto ${port}!`))
```

Ejecutamos la aplicación en el terminal con la línea: node hello-world.js:

```
PS D:\Docencia\UDEA\Desarrollo web\Curso_MERN\CRUD\server> node hello-world.js
La app hello worl esta en el puerto 3001!
```

Teniendo claro cómo funciona, vamos ahora a poner nuestra aplicación a funcionar de manera adecuada, para lo cual necesitaremos de un middleware, que es una especie de extensiones para nuestro servidor Express, que nos ayudará a decodificar el cuerpo de la solicitud HTTP, analizarlo y actuar dependiendo de ello.



Instalaremos body- parser y cors, lo cual hacemos con la siguiente línea en la terminal:

```
npm install --save body-parser
npm install --save cors
```

## Crear (CREATE)

Para desarrollar esta funcionalidad crearemos un archivo denominado libro-api.js y, al igual que en nuestra prueba, importamos el marco de trabajo, y de paso importamos las dos instalaciones que acabamos de hacer de cors y body-parser, creamos la instancia y configuramos el puerto:

```
const express = require('express')
const bodyParser = require('body-parser');
const cors = require('cors');
const path = require('path');

const app = express()
const port = 3000

let libros = []// Aquí ubicaremos los datos de libros

app.use(cors());
app.get('/', function(req, res) {// desplegamos el index
    res.sendFile(path.join(__dirname, '/index.html'));
  });

// Configuración del middleware body parser
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());

app.post('/libro', (req, res) => {
    // Pondremos Código mas Adelante aquí
```

```
});

app.listen(port, () => console.log(`App Libro esta en el puerto ${port}!`));
```

La matriz **libros** será donde almacenaremos la colección de libros.

En la configuración de bodyParser vemos las solicitudes HTTP **urlencoded**, que es un cuerpo predeterminado para formularios, mientras que el .json es lo que usaríamos para solicitar un recurso, sea usando JQuery o en nuestro caso el cliente desde REST y desde el back-end.

Body-parser captura el cuerpo HTTP, decodifica la información y la agrega al archivo req.body. Esto nos permitirá recuperar la información de un libro.

Vamos ahora a agregar la matriz de libros al método app.post:

```
app.post('/libro', (req, res) => {
    const libro = req.body;

    // salida del libro por consola
    console.log(libro);
    books.push(libro);

    res.send('El libro ha sido agregado a la base de datos ');
});
 app.post('/libro', (req, res) => {
    const libro = req.body;

    // salida del libro por consola
    console.log(libro);
    books.push(libro);

    res.send('El libro ha sido agregado a la base de datos ');
```

Ahora crearemos nuestro index.html, donde pondremos nuestro formulario. Aquí estamos importando un estilo bootstrap para darle un poco de vida a nuestra app:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>Crear Nuevo Libro</title>
```

Ahora crearemos nuestro index.html, donde pondremos nuestro formulario. Aquí estamos importando un estilo bootstrap para darle un poco de vida a nuestra app:

```
<!DOCTYPE html>
<html lang="en">
```

<head> Ahora crearemos nuestro index.html, donde pondremos nuestro formulario. Aquí estamos importando un estilo bootstrap para darle un poco de vida a nuestra app:

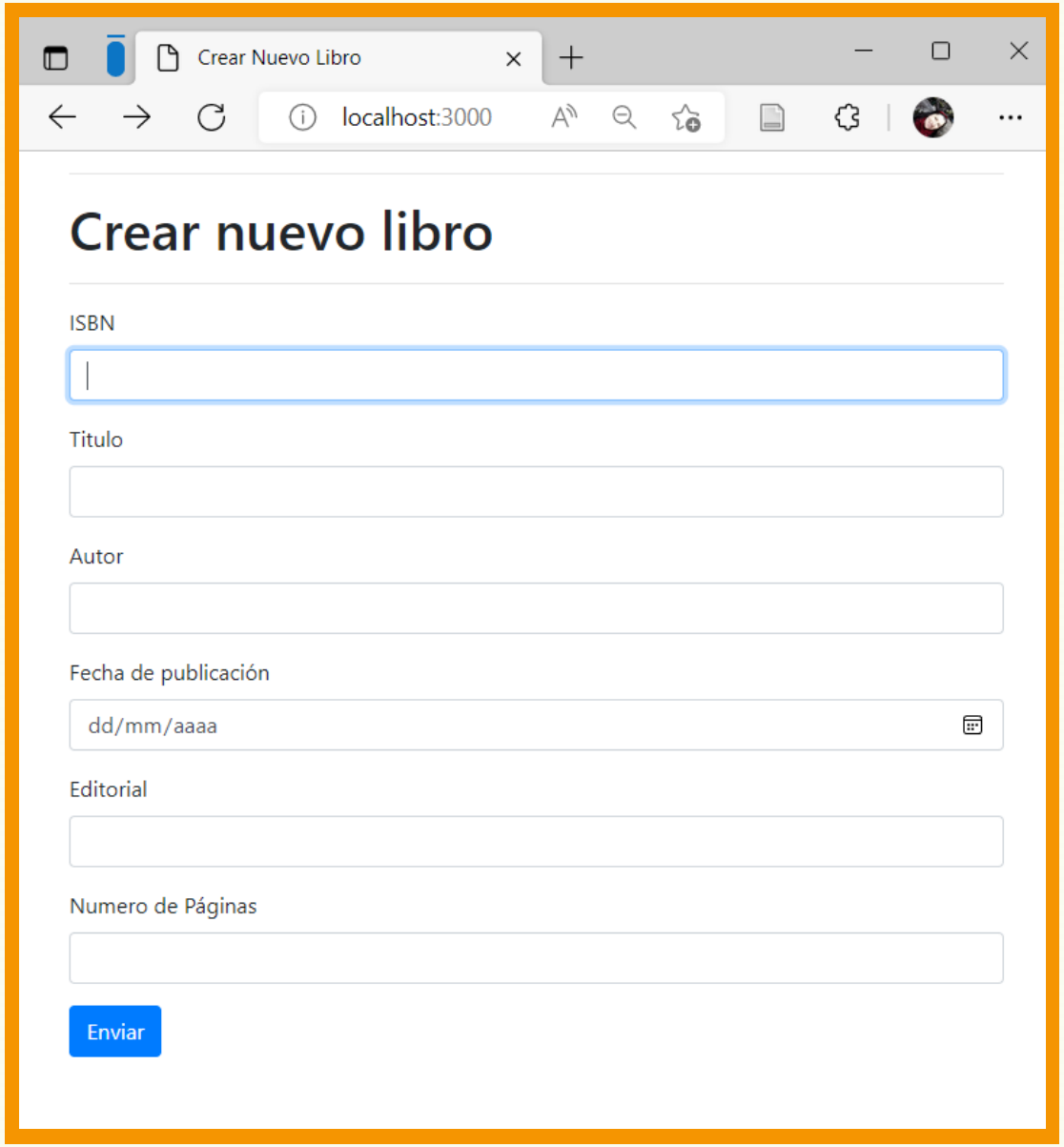
```
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <meta http-equiv="X-UA-Compatible" content="ie=edge">
```

```
    <title>Crear Nuevo Libro</title>


    <link rel="stylesheet"
href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/b
ootstrap.min.css">
</head>


<body>
    <div class="container">
        <hr>
        <h1>Crear nuevo libro</h1>
        <hr>


        <form action="http://localhost:3000/libro"
method="POST">
            <div class="form-group">
                <label for="exampleInputPassword1">ISBN</label>
                <input class="form-control" name="isbn">
            </div>
```

```
            <div class="form-group">
                <label for="exampleInputPassword1">Titulo</label>
                <input class="form-control" name="titulo">
            </div>

            <div class="form-group">
                <label for="exampleInputPassword1">Autor</label>
                <input class="form-control" name="autor">
            </div>

            <div class="form-group">
                <label for="exampleInputPassword1">Fecha de
publicación</label>
                <input type="date" class="form-control"
name="publicacion">
            </div>
```

```
<div class="form-group">
            <label for="exampleInputPassword1">Editor</label>
            <input class="form-control" name="editor">
        </div>

        <div class="form-group">
                <label for="exampleInputPassword1">Numero de
Páginas</label>
                    <input type="number" class="form-control"
name="paginas">
        </div>

                    <button type="enviar" class="btn btn-
primary">Enviar</button>
      </form>
   </div>
</body>
</html>
```

Enviamos nuestra API utilizando **action** en el formulario HTML, en el <form> estará nuestro end-point y la información que enviamos con el botón **enviar** es la información que nuestro método analiza y agrega a la matriz:



## Obtener libros (READ)

Ahora crearemos un end-point para obtener los libros del API. Primero los pondremos directamente en un archivo JSON. Ubícate en la matriz **libros** y agrega un par de libros directamente por código:

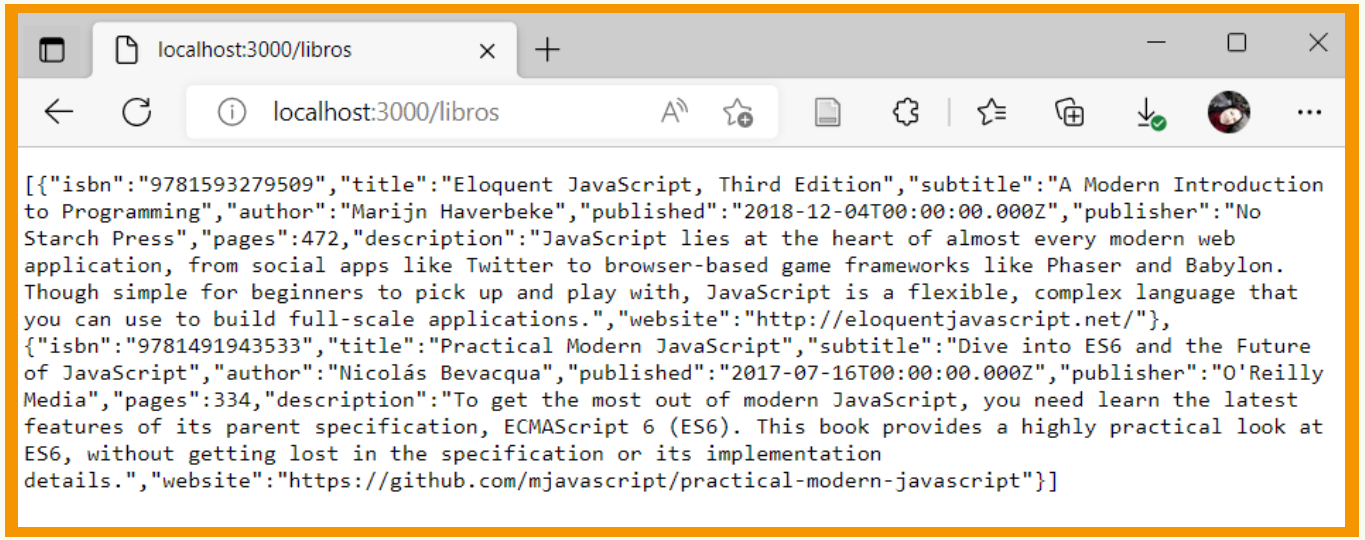
```
let libros = [
  {
    "isbn":"9781593279509",
    "title":"Eloquent JavaScript, Third Edition",
    "subtitle":"A Modern Introduction to Programming",
    "author":"Marijn Haverbeke",
    "published":"2018-12-04T00:00:00.000Z",
    "publisher":"No Starch Press",
    "pages":472,
    "description":"JavaScript lies at the heart of almost every modern web application, from social apps like Twitter to browser-based game frameworks like Phaser and Babylon. Though simple for beginners to pick up and play with,
```

```
JavaScript is a flexible, complex language that you can use to build full-scale applications.",
        "website":"http://eloquentjavascript.net/"
    },
    {
        "isbn":"9781491943533",
        "title":"Practical Modern JavaScript",
        "subtitle":"Dive into ES6 and the Future of JavaScript",
        "author":"Nicolás Bevacqua",
        "published":"2017-07-16T00:00:00.000Z",
        "publisher":"O'Reilly Media",
        "pages":334,
         "description":"To get the most out of modern JavaScript, you need learn the latest features of its parent specification, ECMAScript 6 (ES6). This book provides a highly practical look at ES6, without getting lost in the specification or its implementation details.",
        "website":"https://github.com/mjavascript/practical-modern-javascript"
    }
]/
```

Posteriormente vamos a crear el end-point para mostrarlos:

```
app.get('/libros', (req, res) => {
    res.json(libros);
});
```

Debes visualizarlo como archivo plano:




En este punto ya puedes iniciar mostrando los datos de manera más amigable en un formulario. Sé creativo y continúa tu aplicación.