HERRAMIENTAS PARA CREAR LA APPLICAION

https://carlosazaustre.es/como-crear-una-api-rest-usando-node-js
http://gmoralesc.me/creando-apis-con-node-js-express-y-mongodb-draft-sample.pdf

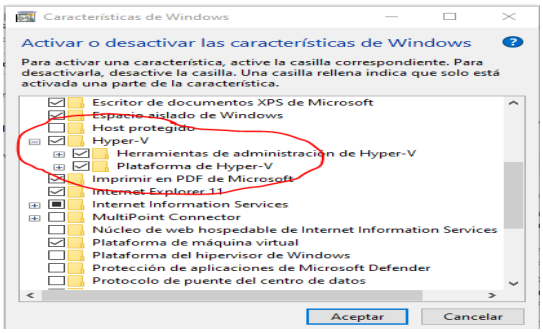
1.- Instalar Node Js

Última version estable.

2.- Trabajar con HYPER-V

https://www.jasoft.org/Blog/post/como-utilizar-hyper-v-y-virtualbox-en-el-mismo-equipo-con-windows-10

Para **poder ejecutar docker** es necesario activar HYPER-V en Windows 10
(Inicio => Buscar **Aplicaciones y características / Programas y Características/Activar o desactivar las caracteísticas dewindows.**



Leer y seguir https://www.jasoft.org/Blog/post/como-utilizar-hyper-v-y-virtualbox-en-el-mismo-equipo-con-windows-10

3.- INSTALAR DOCKER y MONGODB

*Descargar e instalar :***Docker Desktop Installer.exe**
Instalar kitematic(No es necesario de momento).

4.- Crear máquina docker con MongoDb

Instalar MongoDb en Docker
https://platzi.com/tutoriales/1533-mongodb/4930-instalar-mongo-db-usando-docker/

Abrir **PoweShell** como **administrador** y ejecutar el comando
Ejecutar el comando:
docker run -d -p 27017:27017 --name mydatabasee mongo:4.2

5.- NODEJS y NPM

Comprobar versión de node Js
Comprobar versión de npm

6.- INSTALAR VISUAL STUDIO CODE

Desde consola creamos la carpeta del proyecto

mkdir api-rest

Abrir Visual Studio Code

Y desde el menú de la aplicación abrimos la **carpeta del proyecto**.

Abrimos una consola de comandos Menú **Terminal/ new terminal.** Desde este terminal podremos ejecutar los comandos necesarios para crear la aplicación **node js**

7.-BACK END DE LA APPLICAIÓN

7.1 CREAR EL ESQUELETO DE LA APLICACIÓN

Al crear un nuevo proyecto con npm init, se lanzará un asistente que tras algunas preguntas, crea un archivo llamado package.json en la carpeta raíz del proyecto, donde coloca toda la información que se conoce sobre el mismo. Este archivo es un simple fichero de texto, en formato JSON que incorpora a través de varios campos información muy variada

https://lenguajejs.com/npm/administracion/package-json/

Creamos el fichero **package.json** por medio del comando **npm init.**

Al ejecutarse va realizando una serie de preguntas que se reflejan en el fichero

```
{
 "name": "api-rest",
 "version": "1.0.0",
 "description": "Primer aPI REST Full",
  "main": "app.js",
 "scripts": {
  "start": "nodemon app.js",
  "test": "echo \"Error: no test specified\" && exit 1"
 },
 "author": "Fernando",
 "license": "MIT",
 "dependencies": {
  "-": "0.0.1",
  "D": "^1.0.0",
    "body-parser": "^1.19.0",
    "express": "^4.17.1",
    "mongoose": "^5.12.7",
  "nodemon": "^2.0.7"
 }
}
```

El fichero **package.json** es el manifiesto de la aplicación.

7.-2 Instalar express.

Express un framework(entorno de trabajo) que nos facilita la comunicación vía http con el servidor. Estando dentro de la carpeta del proyecto(api-rest) ejecutamos el comando:

npm install express –save

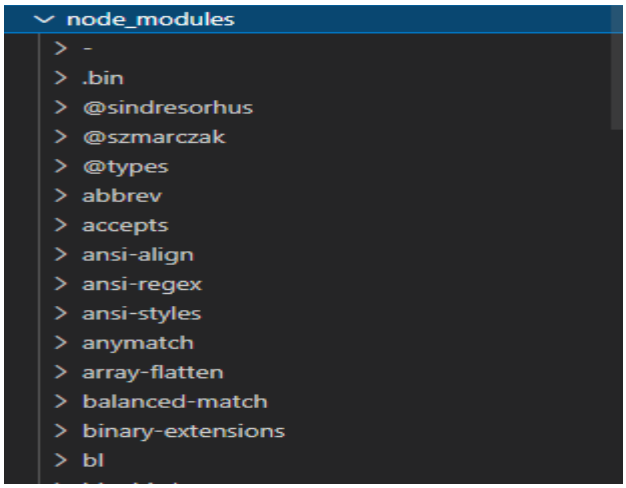
* ***--save*** *actualiza el fichero* ***package.json***

Se crea dentro del proyecto una carpeta **node-modules** , que además de **express** instala todas las librerías que necesitamos para desarrollar una aplicación node js

7.-3 Instalar body-parse

Desde la consola de comandos: **npm install –save body-parse**

Esta librería se encarga de tratar las peticiones http(post, get,...)



8.- FICHEROS de la APLICACIÓN

8.1 Crear la carpeta ***models*** y en ella, el fichero **libro.js**. En este fichero se define el tipo de datos con los que trabaja el script **app.js,** que crea la funcionalidad del Back End. El tipo de dato que ce crea con ***libro.js*** define los campos de la tabla ***Libros*** que se crea en **mongodb**.

```
'use strict'
const mongoose = require('mongoose')
const Schema = mongoose.Schema
const LibrosSchema = Schema({
isbn: String,
titulo: String,
autor: String,
editorial: String,
paginas:Number
})
module.exports = mongoose.model('Libros',LibrosSchema)
```

La base de datos **mongodb,** se crea cuando se ejecute el siguiente código en **app.js**

```
mongose.connect('mongodb://localhost:27017/mydatabasee',(err,res)=>{
  if(err) {
    return console.log("Erroe de conexion ${}")
  }
  console.log('Conexión establecida')
  app.listen(port,()=>{
    console.log(`Api Rest ejecutandose en http:/localhost:${port}`)
  })
})
```

La tabla **Libros** se crea cuando se crea un primer registro.

```
app.post('/api/libro',(req,res)=>{
  //console.log(req.body)
  // res.send({message : 'Producto recibido'})
  console.log('POST /api/libro')
  console.log(req.body)
  let libro = new  Libros()
  libro.isbn= req.body.isbn
  libro.titulo= req.body.titulo
  libro.autor= req.body.autor
  libro.editorial= req.body.editorial
  libro.paginas= req.body.paginas
  libro.save((err,libroStored)=>{
    if(err) res.status(500).send('message : Error al grabar: '+err)
    res.status(200).send({libro:libroStored})
  })
})
```

8.2 Fichero **app.js.**

Es el punto de estrada a la aplicación, podemos llamar a este fichero como queramos, aunque suele ser app.js o index.js. Esto depende del nombre introducido en la variable

```
"main": "app.js"
```

y en

```
"scripts": {
  "start": "nodemon app.js",
```

del fichero **package.json**

Fichero app.js completo.

```
// Carga las librerías necesarias para gestionar el servidor node js con express y el acceso a la
// Base de datos MongoDb con mongoose. También carga el modelo definido en el script libro.js
// Por otra parte define el puerto(3000) donde va a escuchar el servidor node js
// Por medio de Cors( ya hablaremos mas adelante sobre esto) habilitamos que se pueda acceder
// al servidor desde un dominio distinto.
'use static'
const express= require('express')
//const bodyParser = require('body-parser')
const bodyParser = require('body-parser')
const mongose= require('mongoose')
const Libros= require('./models/libro')
const app = express()
const port = process.env.PORT || 3000
const cors = require('cors');
app.use(cors());
//https://stackoverflow.com/questions/36878255/allow-access-control-allow-origin-header-using-html5-fetch-api
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json())


// Lee todos los registros de Libros
app.get('/api/libro',(req,res)=>{

  Libros.find({},(err,libro) =>{
    if(err) res.status(500).send('message : Error al leer: '+err)
```

```
    if(!libro) return res.status(404).send('No existen libros')
    //res.send(200, {products:products})
     res.status(200).send(libro)
  })

})

// Lee un registro de Libros  por el Id
app.get('/api/libro/:libroId',(req,res)=>{
  let libroId=req.params.libroId
  Libros.findById(isbn,(err, libro)=>{
    if(err) res.status(500).send('message : Error al leer: '+err)
    if(!libro) return res.status(404).send('No existe')
    res.status(200).send({libro})
  })
})
// Graba los datos recibidos de un formulario(body) en la bd
app.post('/api/libro',(req,res)=>{
  //console.log(req.body)
  // res.send({message : 'Producto recibido'})
  console.log('POST /api/libro')
  console.log(req.body)

  let libro = new  Libros()
  libro.isbn= req.body.isbn
  libro.titulo= req.body.titulo
  libro.autor= req.body.autor
  libro.editorial= req.body.editorial
  libro.paginas= req.body.paginas
  libro.save((err,libroStored)=>{
    if(err) res.status(500).send('message : Error al grabar: '+err)
    res.status(200).send({libro:libroStored})
  })
})
// Borra registro por el id
app.delete('/api/libro/:isbn',(req,res)=>{
  let libroId=req.params.Isbn
  Libros.findById(libroId,(err, libro)=>{
    if(err) res.status(500).send({message:'Error al borrar : ${err}'})
    libro.remove(err =>{
      if(err) res.status(500).send('message : Error al borrar : '+err)
      res.status(200).send({message: 'Regsitro borrdo : '})
    })
   //  if(!product) return res.status(404).send('No existe')

  })
})
```

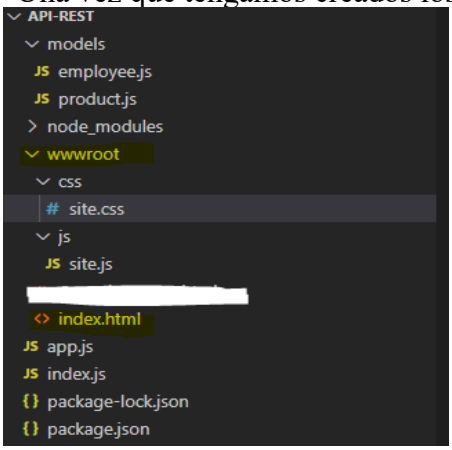
```
// Modifica registro por el id
app.put('/api/libro/:libroId',(req,res)=>{
    let libroId = req.params.libroId
    let registroModificado= req.body;
    Libros.findByIdAndUpdate(libroId,registroModificado, (err,libroUpdated) =>
{
      if(err) res.status(500).send({message: 'Error al modificar: ${err}'})
      res.status(200).send({libro:libroUpdated})
    })
})
// Conexión a la base de datos MongoDb creada en Docker
mongose.connect('mongodb://localhost:27017/mydatabase',(err,res)=>{
    if(err) {
      return console.log("Erroe de conexion ${}")
    }
    console.log('Conexión establecida')
    app.listen(port,()=>{
      console.log("Api Rest ejecutandose en http:/localhost:${port}")
    })
})
```

Qué hace este código app.js? Muy sencillo, las primeras líneas (**require**) se encargan de incluir las dependencias que vamos a usar, algo así como los includes en C o PHP, o los import de Python. Importamos Express para facilitarnos *crear el servidor y realizar llamadas HTTP.* Con http creamos el servidor que posteriormente escuchará en el puerto 3000 de nuestro ordenador (O el que nosotros definamos).

Con **bodyParser** permitimos que pueda *parsear* JSON, methodOverride() nos permite implementar y personalizar métodos HTTP.

Podemos declarar las rutas con ***app.route(nombre_de_la_ruta)*** seguido de los verbos .get(), .post(), etc… y podemos crear una instancia para ellas con express.Router().

Una vez que tengamos creados los ficheros de la aplicación:



Para ponerla en funcionamiento ejecutamos desde el terminal de Visual Studio Code el comando **npm start**

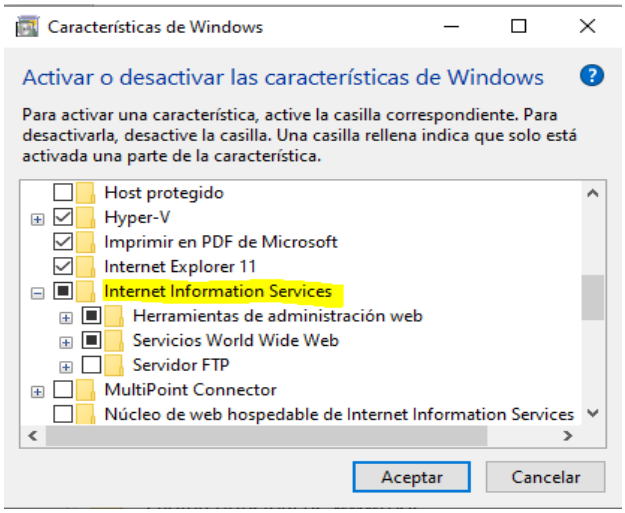
Para ejecutar este código sólo tienes que escribir en consola lo siguiente y abrir un navegador con la url ***http://localhost:3000/api/libro***

FRONT END DE LA APPLICAION

En el Front End se crear la vista de la aplicación Web, el código que se ejecuta en el navegador.

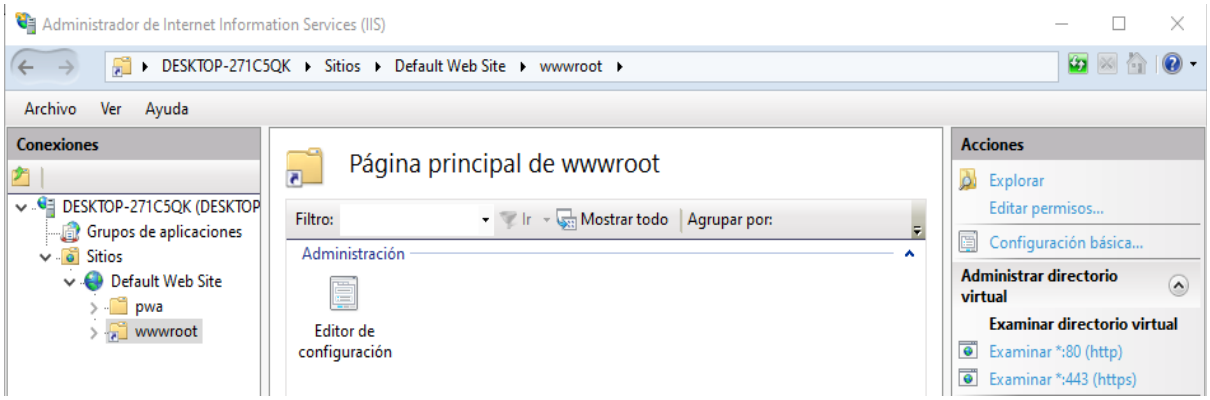
Desde javascript se realizan peticiones AJAX al Back End que realiza las acciones que les solicitemos.

También es necesario instalarlo en un sevidor web. No es necesario que sea el mismo. Para este ejercicio utilizaremos Internet Informatión Server(IIS). Solamente tenemos que Activar Características en Windows 10 y estará trabajando en el puerto 80.



Después creamos una carpeta con los ficheros de la aplicación. El de entrada de la aplicación ha de llamarse index.html

Una vez creada la página ejecutamos IIS y configuramos en Default Web Site , un Sitio Virtual (wwwroot) en el que indicamos la ruta absoluta de la carpeta wwwroot , de la aplicación.



Ficheros del Front End

Fichero con la hoja de estilos **site.css**

```
input[type='submit'], button, [aria-label] {
  cursor: pointer;
}
#editForm {
  display: none;
}
table {
  font-family: Arial, sans-serif;
  border: 1px solid;
  border-collapse: collapse;
}
th {
  background-color: #f8f8f8;
  padding: 5px;
}
td {
  border: 1px solid;
  padding: 5px;
}
```

Fichero con la vista de la pagina html de estilos **index.html**

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>To-do CRUD</title>
  <link rel="stylesheet" href="css/site.css" />
  <script type="text/javascript">
  </script>
</head>
<body>
  <h5> CRUD   https://docs.microsoft.com/es-es/aspnet/core/tutorials/web-api-
javascript?view=aspnetcore-5.0</h5>
  <h3>Add</h3>
  <form action="javascript:void(0);" method="POST" onsubmit="addItem()">
    <table>
      <tr>
        <th>Isbn</th>
        <th>Titulo</th>
        <th>Autor</th>
        <th>Editorial</th>
        <th>Paginas</th>
      </tr>
      <tr>
        <th>
          <input type="text" id="add-isbn">
```

```
        </th>
        <th>
          <input type="text" id="add-titulo">
        </th>
        <th>
          <input type="text" id="add-autor">
        </th>
        <th>
          <input type="text" id="add-editorial">
        </th>
        <th>
          <input type="text" id="add-paginas">
        </th>
      </tr>
    </table>
    <input type="submit" value="Añadir registro">
  </form>
  <div id="editForm">
    <h3>Edit</h3>
    <form action="javascript:void(0);" onsubmit="updateItem()">
      <input type="hidden" id="edit-id">
      <input type="text" id="edit-isbn">
      <input type="text" id="edit-titulo">
      <input type="text" id="edit-autor">
      <input type="text" id="edit-editorial">
      <input type="text" id="edit-paginas">
      <input type="submit" value="Save">
      <a onclick="closeInput()" aria-label="Close">&#10006;</a>
    </form>
  </div>

  <p id="counter">N  de Registros</p>

  <table>
    <tr>
      <th>Isbn</th>
      <th>Titulo</th>
      <th>Autor</th>
      <th>Editorial</th>
      <th>Paginas</th>
      <th></th>
      <th></th>
    </tr>
    <tbody id="todos"></tbody>
    <input type="button" onclick="getItems()" value="Leer"></input>
  </table>
  <script src="js/site.js" asp-append-version="true"></script>
  <script type="text/javascript">
  </script>
</body>
</html>
```

CRUD https://docs.microsoft.com/es-es/aspnet/core/tutorials/web-api-javascript?view=aspnetcore-5.0

Add

Isbn	Titulo	Autor	Editorial	Paginas

Añadir registro

Edit

7 | 8 | 9 | 0 | 11 | Save ✕

2 Registros

Leer

Isbn	Titulo	Autor	Editorial	Paginas		
1	2	3	4	6	Edit	Delete
7	8	9	0	11	Edit	Delete

Fichero con el código javascript que realiza la función de controlador site.js

```
const uri = 'http://127.0.0.1:3000/api/libro';
let todos = [];

function getItems()
{
  fetch("http://127.0.0.1:3000/api/libro/")
    .then(response => response.json())
    .then(data => {  _displayItems(data); })
    .catch(error => console.error('Unable to get items.', error));

}

function addItem() {

  const item = {
    isbn: document.getElementById('add-isbn').value.trim(),
    titulo: document.getElementById('add-titulo').value.trim(),
    autor: document.getElementById('add-autor').value.trim(),
    editorial: document.getElementById('add-editorial').value.trim(),
    paginas: document.getElementById('add-paginas').value.trim()
  };



  fetch(uri, {
    method: 'POST',
    headers: {
      'Accept': 'application/json',
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(item)
  })
    .then(response => response.json())
    .then(() => {
      getItems();

    })
    .catch(error => console.error('Error al añadir registro.', error));
}

function deleteItem(id) {
  console.log("ID    "+id)
  fetch(`${uri}/${id}`, {
    method: 'DELETE'
  })
    .then(() => getItems())
    .catch(error => console.error('Error al eliminar el registro.', error));
}
```

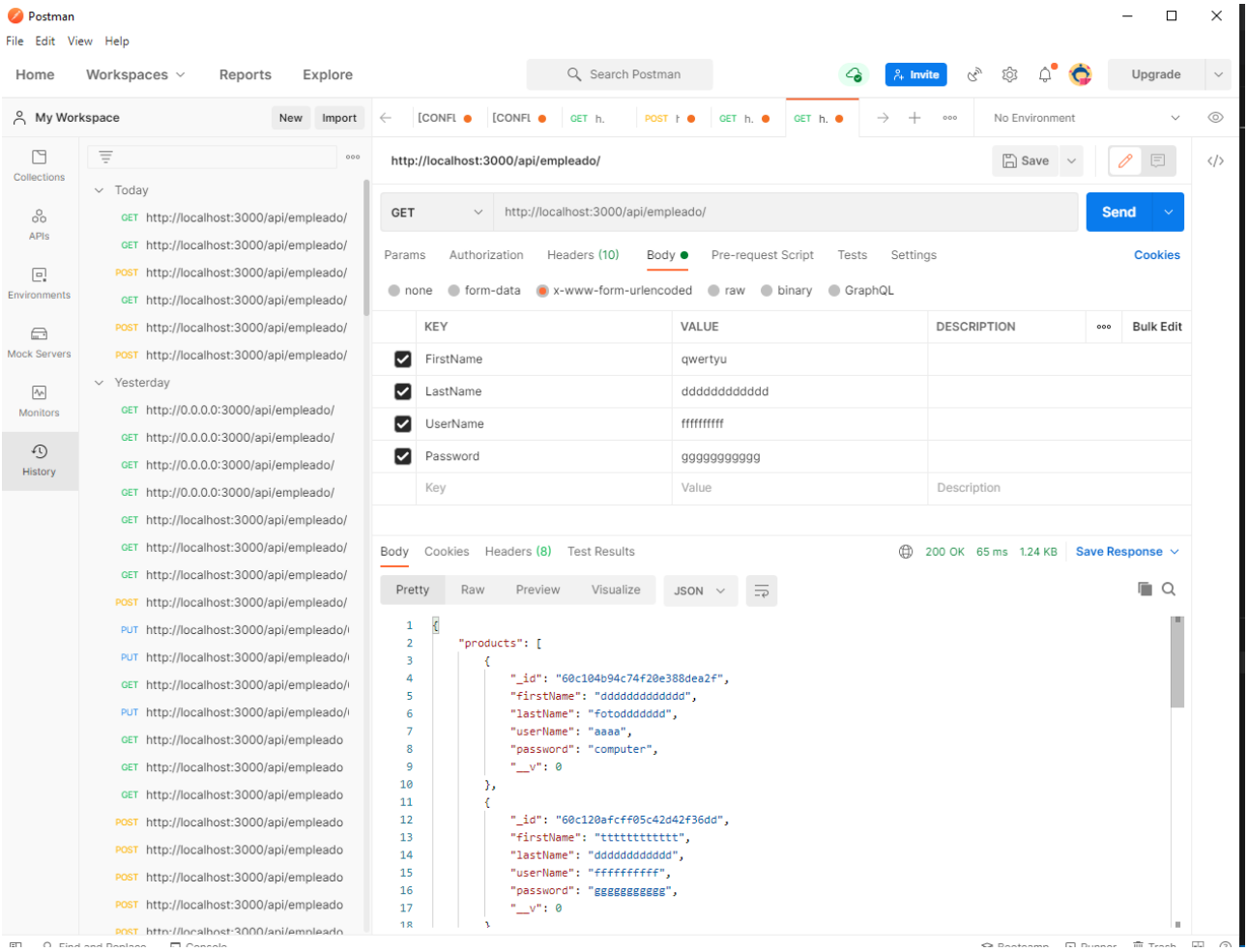
```
function displayEditForm(registro) {
  //Reecibo el registro que quiero editar

  document.getElementById('edit-id').value = registro._id;
  document.getElementById('edit-isbn').value = registro.isbn;
  document.getElementById('edit-titulo').value = registro.titulo;
  document.getElementById('edit-autor').value = registro.autor;
  document.getElementById('edit-editorial').value = registro.editorial;
  document.getElementById('edit-paginas').value = registro.paginas;
  document.getElementById('editForm').style.display = 'block';
}

function updateItem() {
  const itemId = document.getElementById('edit-id').value;
  const item = {
    id: parseInt(itemId, 10),
    isbn: document.getElementById('edit-isbn').value.trim(),
    titulo: document.getElementById('edit-titulo').value.trim(),
    autor: document.getElementById('edit-autor').value.trim(),
    editorial: document.getElementById('edit-editorial').value.trim(),
    paginas: document.getElementById('edit-paginas').value.trim()
  };

  // Envia con el método PUT el registro a modificar en formato JSON
  fetch(`${uri}/${itemId}`, {
    method: 'PUT',
    headers: {
      'Accept': 'application/json',
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(item)
  })
    .then(() => getItems())
    .catch(error => console.error('No se actualiza el registro', error));

  closeInput();

  return false;
}

function closeInput() {
  document.getElementById('editForm').style.display = 'none';
}

function _displayCount(itemCount) {
  const name = (itemCount === 1) ? 'Registro ' : 'Registros ';

  document.getElementById('counter').innerText = `${itemCount} ${name}`;
}
```

```
function _displayItems(data) {

  const tBody = document.getElementById('todos');
  tBody.innerHTML = '';

  _displayCount(data.length);

  const button = document.createElement('button');

  data.forEach(item => {

    let editButton = button.cloneNode(false);
    editButton.innerText = 'Edit';
   //  editButton.setAttribute('onclick', `displayEditForm(${item.id})`);

    // Otra forma mas estandar de llamar a una function y pasarle un  parameto
    // el parametro que pasa item  , tiene los datos de todo el registro del
empleado

    editButton.addEventListener("click", function () { displayEditForm(item ) },
false);

    let deleteButton = button.cloneNode(false);
    deleteButton.innerText = 'Delete';
   // deleteButton.setAttribute('onclick', `deleteItem(${item.id})`);
    // Me gusta mas utilizar eventos con addEventListener que atributos
    // Pasamos el id del registro a eliminar
    deleteButton.addEventListener("click", function () { deleteItem(item._id) },
false);

    let tr = tBody.insertRow();

    let td1 = tr.insertCell(0);
    let textNode = document.createTextNode(item.isbn);
    td1.appendChild(textNode);

    let td2 = tr.insertCell(1);
    let textNode1 = document.createTextNode(item.titulo);
    td2.appendChild(textNode1);

    let td3 = tr.insertCell(2);
    let textNode2 = document.createTextNode(item.autor);
    td3.appendChild(textNode2);

    let td4 = tr.insertCell(3);
    let textNode3 = document.createTextNode(item.editorial);
    td4.appendChild(textNode3);

    let td5 = tr.insertCell(4);
    let textNode4 = document.createTextNode(item.paginas);
```

```
    td5.appendChild(textNode4);

    let td6 = tr.insertCell(5);
    td6.appendChild(editButton);

    let td7 = tr.insertCell(6);
    td7.appendChild(deleteButton);
  });

  todos = data;
  //*************************************************************************
  // Nada mas cargar el fichero js en memoria ejecuta este comando que viosualiza
en una
  // table los resgsitros de empleados obtenidos desde le ApiRest
 // getItems();
}
```

PROBAR EL FUNCIONAMIENTO DE LA APLICACIÓN

Para probar el funcionamiento del Back End, y antes de crear el Front End, podemos instalar y utilizar la herramienta POSTMAN. Que proporciona un interfaz con los que ejecutar los métodos del Api Rest. GET, POST, DELETE y UPDATE.



COMANDOS PARA TRABAJAR CON MONGODB DESDE POWESHELL

Crear máquina docker con MongoDb

Abrir PoweShell como administrador y ejecutar el comando
docker run -d -p 27017:27017 --name mydatabasee mongo:4.2

How To Run MongoDB as a Docker Container – BMC Software | Blogs

```
docker exec -it mydatabase bash
 mongo
 use mydatabasee
db.createCollection("Employee")
 db.Employee.insertMany([ {FirstName: "dos", LastName: "dos2", UserName: "dos3",
Password:"dos4",Campo4:4}])
db.Employee.find().pretty()
```