



¿Qué pasa con la codificación?

Hogar > secuencia de comandos java > Aplicación Full Stack usando Node Js, Express Js, SQL, Bootstrap y Java Script

Aplicación Full Stack usando Node Js, Express Js, SQL, Bootstrap y Java Script

Por Himanshu Shekhar 29 de julio de 2022 3 comentarios

Es posible que haya visto muchos blogs y vídeos que explican cómo crear una aplicación utilizando MERN, MEAN o algún otro marco. Pero, ¿qué sucede si desea crear una aplicación **sin utilizar ningún marco de interfaz o sin utilizar MongoDB**? Entonces, en este caso, los recursos en Internet son limitados o, francamente, muy, muy pocos.

EMPEZAR

1) Haga clic en "Empezar"
|
2) Iniciar la instalación
|
3) Bloquear anuncios y ...

Web Companion

Y es por eso que he seleccionado este artículo donde le mostraré paso a paso (con código) cómo puede crear una aplicación simple y completa usando:

- Nodo JS, Express JS (**backend**)
- SQL (**para base de datos**)
- Java Script, Boostrap 5 (**para interfaz**)

Antes de seguir leyendo el artículo, debe tener un conocimiento básico de las pilas de tecnología mencionadas anteriormente.

Tabla de contenido



Requisito previo

Paso 1: iniciar un servidor node js

Paso 2: crea tu primera API
servidor.js

Paso 3: crear una conexión de base de datos

Paso 4: conectar la base de datos con el backend del nodo (usando el archivo .env)

Paso 5: configuremos la interfaz ahora
res.render()



```
res.redirect()  
res.enviarArchivo()  
Paso 6: crear operación  
Paso 7: leer la operación  
Paso 8 – Eliminar operación  
Paso 9 – Operación de actualización  
Conclusión
```

Requisito previo

- Instalar el nodo js

Vaya a cmd y escriba "nodo -v" para verificar si está instalado o no.

- Instalar el editor de código VS
- Conocimientos básicos de Node JS, Java Script, HTML y CSS.

Mira el video en su lugar

NodeJs and MySQL CRUD operation in Hindi | Step by Step



Ahora que eso está fuera del camino, comenzemos.

Paso 1: iniciar un servidor node js

Cree una carpeta y abra el código VS dentro de ella y escriba el siguiente comando

```
npm init
```

Ahora es posible que estés viendo un archivo package.json creado.

¡Bien! Ahora instalaremos algunas dependencias importantes.



npm install mysql express nodemon dotenv cors body-parser ejs

- **mysql**: el módulo MySQL ayuda a crear conexiones entre el servidor Node.js con la base de datos MySQL.
- **express**: Express.js es un marco de aplicación web para Node.js. Proporciona varias funciones que hacen que el desarrollo de aplicaciones web sea rápido y sencillo, lo que de otro modo llevaría más tiempo utilizando solo Node.js.
- **nodemon**: ayuda en el inicio automático del servidor
- **cors**: resolviendo el problema entre navegadores
- **body-parser**: analiza la solicitud proveniente del cuerpo
- **ejs** – motor de plantillas

```
{
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "body-parser": "^1.20.0",
    "cors": "^2.8.5",
    "dotenv": "^16.0.1",
    "express": "^4.18.1",
    "mysql": "^2.18.1",
    "nodemon": "^2.0.19"
  }
}
```

Ahora verifique en su archivo package.json si todas las dependencias están instaladas o no. Una vez instalado, pasemos al segundo paso.

Paso 2: crea tu primera API

API básicamente significa un conjunto de funciones/lógica/reglas creadas y a las que se puede acceder mediante una URL. (la definición no está completa, pero por ahora vamos a arreglarnos con esto)

Al realizar npm init, se debe haber creado index.js o server.js. Si no, sigue adelante y crea uno.

Comencemos a escribir algo de código ahora.

servidor.js

```

1. const expreso = requerir ("expreso"); //paquete expresivo iniciado
2. aplicación constante = expresar (); // la instancia expresa ha sido creada y se accederá mediante l
3.
4. aplicación. obtener ("/", (solicitud, res) => {
5.   res. enviar ("API en ejecución");
6. });
7.
8. aplicación. escuchar (3000);

```

Además, no olvide agregar nodemon en el archivo package.json.



```

 6     "scripts": {
 7       "start": "nodemon server.js",
 8       "test": "echo \"Error: no test specified\" && exit 1"
 9     },
10     "author": "",
11     "license": "ISC",
12     "dependencies": [
13       ...
14     ]
15   }
16 }

```

Ahora ve a nuestra terminal vs code y escribe

npm start

Vaya al navegador y escriba "http://localhost:3000/"

¡Y bum! has creado tu primera API.

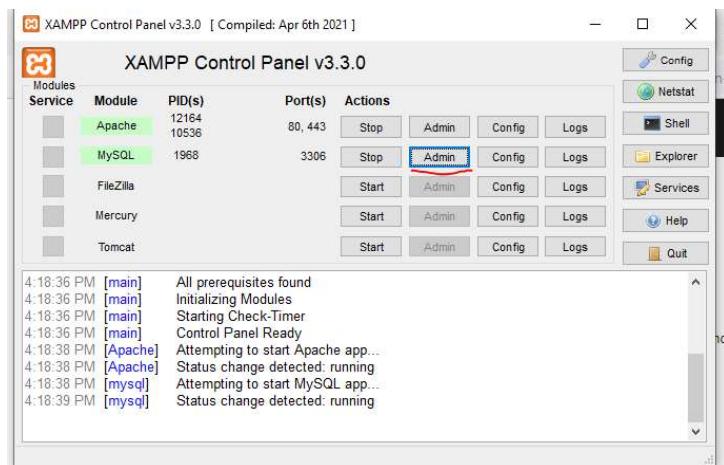


API running

Paso 3: crear una conexión de base de datos

Ahora, para las bases de datos, usamos MySQL. Para ejecutar el servidor MySQL en su local necesitamos instalar xampp

- Instalar Xampp
- Inicie MySQL y el servidor Apache
- Vaya a "http://localhost/phpmyadmin/"



Ahora continúa y crea una base de datos.

Por ejemplo: nombre de la base de datos – practise_database



The screenshot shows the phpMyAdmin interface for a MySQL server at 127.0.0.1. The 'Databases' tab is selected. A red circle highlights the 'Create database' input field where 'practise_database' is typed. An arrow points from the 'Create' button to the right.

Ahora que se creó la base de datos, necesita crear una tabla dentro de la base de datos.

Por ejemplo – nombre de la tabla – demo_table

The screenshot shows the 'Create table' dialog in phpMyAdmin. The 'Name:' field contains 'demo_table', which is circled in red. An arrow points from the 'Go' button to the right.

Ahora agregue una columna en la tabla y cree una estructura de tabla.

Por ejemplo,

col 1, id,

col 2, nombre,

col 3, correo electrónico

The screenshot shows the 'Structure' tab for the 'demo_table'. The table name is 'demo_table'. It contains three columns: 'id' (INT, 10, None, InnoDB), 'name' (VARCHAR, 50, None, InnoDB), and 'email' (VARCHAR, 50, None, InnoDB). The 'Save' button at the bottom is highlighted with a red arrow.

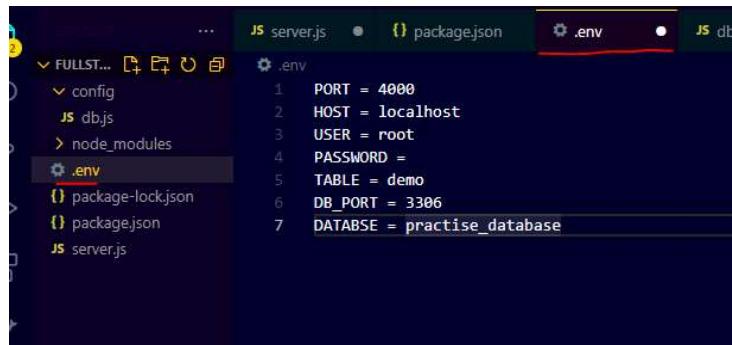
Paso 4: conectar la base de datos con el backend del nodo (usando el archivo .env)

Crear un archivo .env



archivo .env

1. PUERTO = 4000
2. ANFITRIÓN = servidor local
3. USUARIO = raíz
4. CONTRASEÑA =
5. TABLA = demostración
6. PUERTO_DB = 3306
7. BASE DE DATOS = base de datos_practica



Note - The above USER and PASSWORD are by default. You can go to priviledge section of myPhpAdmin and change the password accordingly.

Cree una nueva carpeta con el nombre "config" y un nuevo archivo con el nombre "db.js"

Ruta - crud-app/server/config/db.js

```

1. const dotenv = requerir ("dotenv");
2. const mysql = requerir ("mysql");
3. dotenv.configuración ();
4.
5. conexión constante = mysql.crear conexión ({
6. anfitrión: proceso. env . ANFITRIÓN ,
7. usuario: proceso. env . USUARIO ,
8. contraseña: proceso. env . CONTRASEÑA ,
9. base de datos: proceso. env . BASE DE DATOS ,
10. });
11.
12. conexión. conectar (( error ) => {
13. si ( error ) devuelve la consola. registro ( error );
14. consola. log ("conexión exitosa");
15. });
16.
17. módulo. exportaciones = conexión;
  
```

Ahora actualice su **archivo server.js**

```

1. const expreso = requerir ("expreso"); //paquete expreso iniciado
2. const app = express(); // express instance has been created and will be access by app variable
3. const cors = require("cors");
4. const dotenv = require("dotenv");
5.
  
```

```

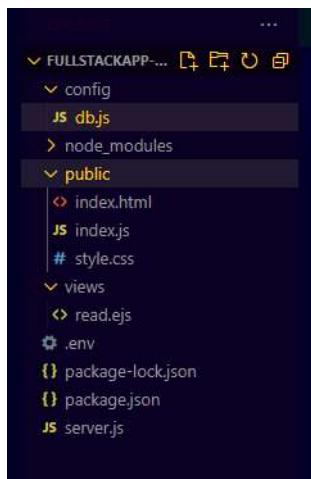
6. const connection = require("./config/db");
7. dotenv.config();
8.
9. app.use(cors());
10. app.use(express.urlencoded({ extended: false }));
11. app.use(express.json());
12.
13. app.get("/", (req, res) => {
14.   res.send("API running");
15. });
16.
17.
18. app.listen(process.env.PORT, function (err) {
19.   if (err) console.log(err);
20.   console.log(`listening to port ${process.env.PORT}`);
21. });

```

Step 5 – Let's setup frontend now

Enough of the backend, let's create some UI now.

Create a folder “public” and file name “index.html”– **crud-app/server/public/index.html**



index.html

```

1. <!DOCTYPE html>
2. <html lang="en">
3.   <head>
4.     <meta charset="UTF-8" />
5.     <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6.     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7.     <link
8.       href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0/dist/css/bootstrap.min.css"
9.       rel="stylesheet"
10.      integrity="sha384-gH2yIjqKdNHPEq0n4Mqa/HGKIhSkIHeL5AyhkYV8i59U5AR6csBvApHHNI/vlIBx"
11.      crossorigin="anonymous"
12.    />
13.    <link rel="stylesheet" href="style.css" />
14.    <title>Document</title>
15.  </head>
16.  <body>
17.    <div class="container my-2">
18.      <h1 class="text-center">Fill the data</h1>
19.      <form id="myForm" action="http://localhost:4000/create" method="post">
20.        <div class="mb-3">

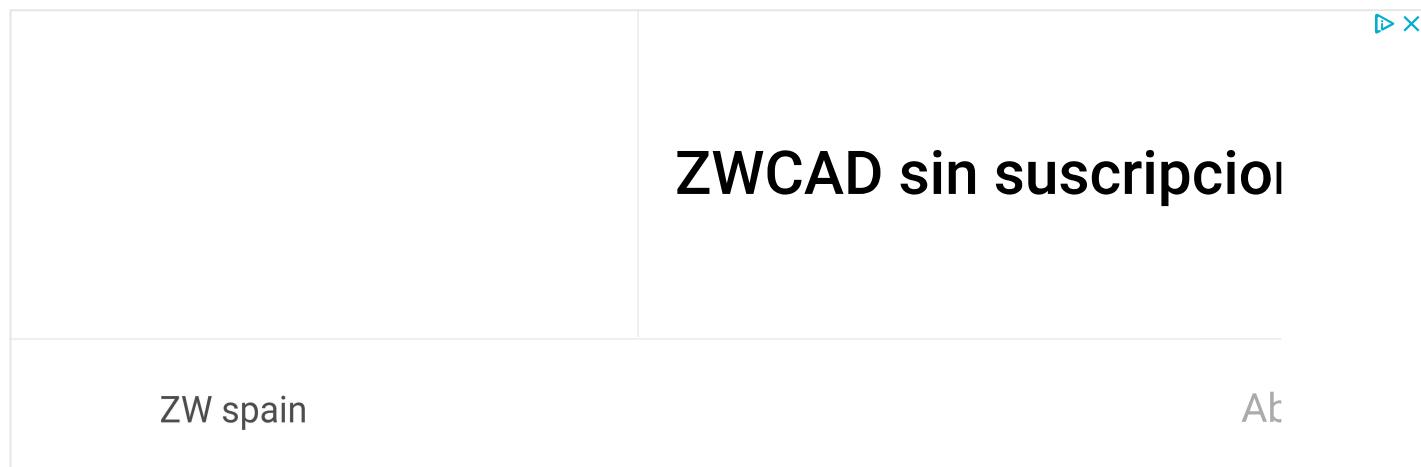
```

```

21.   <label class="form-label">Name</label>
22.   <input type="text" class="form-control" name="name" />
23.   </div>
24.   <div class="mb-3">
25.     <label class="form-label">Email address</label>
26.     <input type="email" class="form-control" name="email" />
27.   </div>
28.
29.   <button type="submit" class="btn btn-primary">Submit</button>
30. </form>
31. </div>
32. </body>
33. </html>

```

Now that we have created our form UI, let's render it and show it to the browser (This is noting but read operation. Anyways we will be seeing read in details as you proceed in the article)



Now we need to create one router with "/" and render an HTML file which is **index.html** in our case

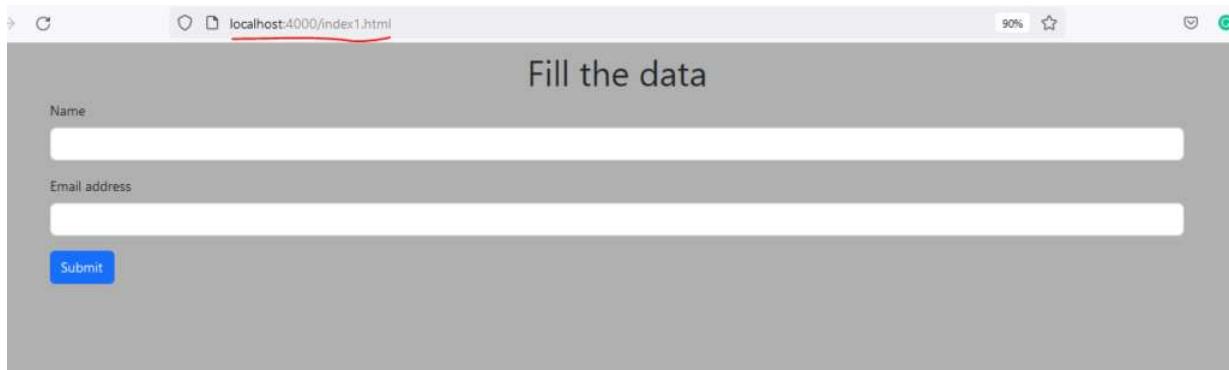
server.js

```

1. const express = require("express"); //express package initiated
2. const app = express(); // express instance has been created and will be access by app variable
3. const cors = require("cors");
4. const dotenv = require("dotenv");
5.
6. const connection = require("./config/db");
7. dotenv.config();
8.
9. app.use(cors());
10. app.use(express.urlencoded({ extended: false }));
11. app.use(express.json());
12.
13.
14. //read file on UI
15. app.get("/", (req, res) => {
16.   res.redirect("/index.html");
17. });
18.
19.
20. app.listen(process.env.PORT, function (err) {
21.   if (err) console.log(err);
22.   console.log(`listening to port ${process.env.PORT}`);
23. });

```

Now go to <http://localhost:4000> and BOOM our UI is rendered.



Now before you start having confusion between the below 3, read the difference now itself.

res.render()

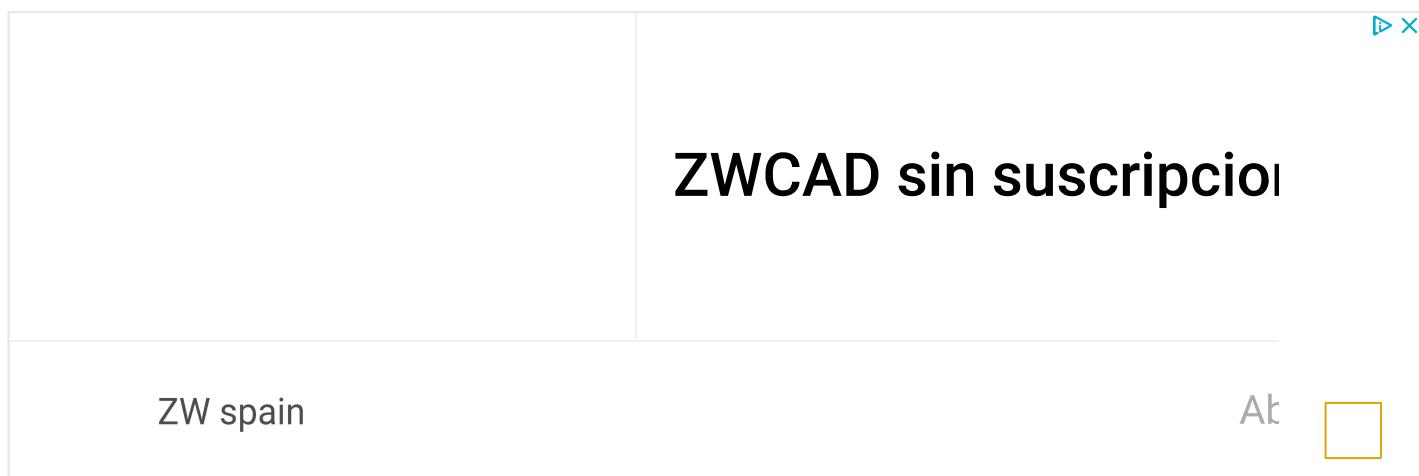
Render means you are collecting some information through a data set. means you go to Mumbai and ask for some good places to explore that is data rendering. and in computer terms, you set a function to get the location of a particular user when he comes to a particular page. then this is the data rendering of the location

res.redirect()

Redirect means you are sending someone to a particular position. for example, if you are going to Pune to Mumbai then you are redirected to Pune to Mumbai. in computer terms if you click on any link then it will redirect you on a specific page. that is called redirect.

res.sendFile()

El método sendfile, por otro lado, simplemente envía un archivo determinado al cliente, independientemente del tipo y contenido del archivo.



Paso 6: crear operación

Así que ahora, nuestro objetivo es que cuando el usuario ingrese el nombre y el correo electrónico y presione el botón Enviar, debemos hacer 3 cosas:

1. Presione el enrutador en el archivo server.js
2. Guarde los datos en la base de datos.
3. Redirigir al usuario a otra página donde pueda ver todo lo ingresado.

Bueno, el siguiente código hará los 3 trabajos:

```

1. //crear
2. aplicación. publicar ( "/crear" , ( req, res ) => {
3.   consola. log ( solicitud. cuerpo . nombre );
4.   nombre var = solicitud. cuerpo . nombre ;
5.   var correo electrónico = req. cuerpo . correo electrónico ;
6.   intentar {
7.     conexión. consulta (
8.       "INSERTAR en los valores demo123 (nombre, correo electrónico) (??)" , //2. guardar en base de dc
9.       [ nombre Correo Electronico ],
10.      función ( errar, resultado ) {
11.        si ( errar ) {
12.          consola. iniciar sesión ( errar );
13.        } demás {
14.          // res.json({ resultado });
15.          res. redirigir ( "/datos" ); //3. Redirigir al usuario a la nueva página
16.        }
17.      }
18.    );
19.  } atrapar ( errar ) {
20.   res. enviar ( errar );
21. }
22. });

```

[Raw](#)
[Copy](#)
[Extern](#)

```

<!-- hitting the /create router -->
<form id="myForm" action="http://localhost:4000/create" method="post">
  <div class="mb-3">
    <label class="form-label">Name</label>
    <input type="text" class="form-control" name="name" />
  </div>
  <div class="mb-3">
    <label class="form-label">Email address</label>
    <input type="email" class="form-control" name="email" />
  </div>

```

Esta acción afectará al enrutador "/create"

Servidor final.js

```

1. const expres = requerir ( "expreso" ); //paquete expreso iniciado
2. aplicación constante = expresar () ; // la instancia expresa ha sido creada y se accederá mediante l
3. const cors = requerir ( "cors" );
4. const dotenv = requerir ( "dotenv" );
5. var bodyParser = require ( "body-parser" );
6. conexión constante = requerir ( "./config/db.js" );
7.
8. dotenv. configuración ();

```



```

9. aplicación.use(bodyParser.urlencoded({ extendido: verdadero }));
10. aplicación.set("motor de vista", "ejs");
11. aplicación.utilizar(cors());
12. aplicación.utilizar(expreso.json());
13. aplicación.uso(express.static(__dirname + "/public"));
14. aplicación.uso(expreso.estático(__dirname + "/vistas"));
15.
16. //leyendo el formulario
17. aplicación.obtener("/", (solicitud, res) => {
18.   res.redirigir("/index1.html");
19. });
20.
21.
22.
23. //crear
24. aplicación.publicar("/crear", (req, res) => {
25.   consola.log(solicitud.cuerpo.nombre);
26.   nombre = solicitud.cuerpo.nombre;
27.   var correoElectrónico = req.cuerpo.correoElectrónico;
28.   intentar {
29.     conexión.consulta(
30.       "INSERTAR en los valores demo123 (nombre, correo electrónico) (?,?)",
31.       [nombre Correo Electronico],
32.       función(errar, resultado) {
33.         si(errar) {
34.           consola.iniciar sesión(errar);
35.         } demás {
36.           // res.json({resultado});
37.           res.redirigir("/datos");
38.         }
39.       }
40.     );
41.   } atrapar(errar) {
42.     res.enviar(errar);
43.   }
44. });
45.
46. aplicación.escuchar(proceso.entorno.PUERTO || 3000, función(err) {
47.   si(err) consola.iniciar sesión(errar);
48.   consola.log(`escuchando el puerto ${process.env.PORT}`);
49. });

```

Paso 7: leer la operación

Ahora, una vez que hayamos enviado el formulario en la base de datos, debemos redirigir al usuario a una página donde pueda ver los datos que ha ingresado, ¿verdad?

Sí significa que necesitamos añadir un valor dinámicamente a una página HTML. Y ahí es donde entra en juego [el motor de plantillas](#).

ZWCAD sin suscripción

ZW spain

Añadir

Ahora hay muchos motores de plantillas como HUF, haml, hbs, ejs, etc., pero en este artículo usaremos [el motor de plantillas ejs](#).

Keeping the article length in mind, please click on the hyperlink and read more about templating engine yourself.

In short -

use this to

<% write java script inside me %>

and this

<%= %> or <%= %> to write HTML string

Tags

- <% 'Scriptlet' tag, for control-flow, no output
- <%_ 'Whitespace Slurping' Scriptlet tag, strips all whitespace before it
- <%= Outputs the value into the template (HTML escaped)
- <%- Outputs the unescaped value into the template
- <%# Comment tag, no execution, no output
- <%> Outputs a literal '<%'
- %> Plain ending tag
- -%> Trim-mode ('newline slurp') tag, trims following newline
- _%> 'Whitespace Slurping' ending tag, removes all whitespace after it

Empecemos.

Antes de crear un archivo ejs, necesitamos crear un enrutador de operación de lectura que se activará cuando el enrutador de operación de creación haya hecho su trabajo y envíe datos al archivo ejs. El motor de plantillas consumirá datos y los mostrará en la interfaz de usuario simultáneamente.

No lo entendí, espera.



El siguiente fragmento de código no es más que una operación de lectura que:

- leer todos los datos de la base de datos
- enviar todos los datos al motor de plantillas, ejs en nuestro caso

```

1. // operación de lectura que pasará valor al motor ejs
2. aplicación. obtener ( "/datos" , ( solicitud, res ) = > {
3.   const allData = "seleccionar * de demo123";
4.   conexión. consulta ( todos los datos, ( err, filas ) = > {
5.     si ( err ) {
6.       res. enviar ( err );
7.     } demás {
8.       // res.json({ filas });
9.       res. render ( "read.ejs" , { filas } ); // volver a escribir el archivo read.ejs junto con los datos
10.    }
11.  });
12. });

```

Cree un nombre de carpeta "vistas" y un nombre de archivo "**read.ejs**" (ruta: **crud-app/server/views/read.ejs**)

leer.ejs

```

1. <!DOCTYPE html>
2. <html lang= "es" >
3.   <cabeza >
4.     <meta juego de caracteres = "UTF-8" />
5.     <meta http-equiv= "X-UA-Compatible" content= "IE=edge" />
6.     <meta nombre = "ventana gráfica" contenido = "ancho = ancho del dispositivo, escala inicial = 1,0"
7.     <enlace
8.       href= "https://cdn.jsdelivr.net/npm/bootstrap@5.2.0/dist/css/bootstrap.min.css"
9.       rel= "hoja de estilo"
10.      integridad = "sha384-gH2yIJqKdNHPEq0n4Mqa/HGKIhSkIHeL5AyhkYV8i59U5AR6csBvApHHNI/vlIBx"
11.      origen cruzado = "anónimo"
12.    />
13.
14.    <enlace rel= "hoja de estilo" tipo= "texto/css" href= "estilo.css" />
15.    <título > Documento </título >
16.  </cabeza >
17.  <cuerpo >
18.    <clase div = "contenedor" >
19.      <h1 class = "h3 text-center" > Todos los datos </h1 >
20.      <clase div = "fila" >
21.        <% si ( filas. longitud > 0 ) { %> <% filas. mapa (( cadaData ) = > { %>
22.          <div clase = "col-md-3 col-sm-12 mx-1" >
23.            <clase div = "tarjeta" estilo = "ancho: 18rem" >
24.              <clase div = "cuerpo de tarjeta" >
25.                <h5 class = "título de la tarjeta" ><%- eachData. nombre %></h5 >
26.                <h6 class = "tarjeta-subtítulo mb-2 texto-silenciado" >
27.                  <%- cada dato. correo electrónico %>
28.                </h6>
29.                <a href="#" class="card-link">Edit</a>
30.                <a href="#" class="card-link">Delete</a>
31.              </div>
32.            </div>
33.          </div>
34.
35.        <% } ) %> <%} else { %>
36.          <div clase="col-md-12" >
37.            <h1 class="h2 text-center" >No data found</h1>
38.          </div>
39.        
```



```

40.    <% }%>
41.    </div>
42.    </div>
43.    </body>
44.    </html>

```

Updating **server.js** file

```

1. const express = require("express"); //express package initiated
2. const app = express(); // express instance has been created and will be access by app variable
3. const cors = require("cors");
4. const dotenv = require("dotenv");
5. var bodyParser = require("body-parser");
6. const connection = require("./config/db.js");
7.
8. dotenv.config();
9. app.use(bodyParser.urlencoded({ extended: true }));
10. app.set("view engine", "ejs");
11. app.use(cors());
12. app.use(express.json());
13. app.use(express.static(__dirname + "/public"));
14. app.use(express.static(__dirname + "/views"));
15.
16.
17. app.get("/", (req, res) => {
18.   res.redirect("/index1.html");
19. });
20.
21. //read
22. app.get("/data", (req, res) => {
23.   const allData = "select * from demo123";
24.   connection.query(allData, (err, rows) => {
25.     if (err) {
26.       res.send(err);
27.     } else {
28.       // res.json({ rows });
29.       res.render("read.ejs", { rows });
30.     }
31.   });
32. });
33.
34. //create
35. app.post("/create", (req, res) => {
36.   console.log(req.body.name);
37.   var name = req.body.name;
38.   var email = req.body.email;
39.   try {
40.     connection.query(
41.       "INSERT into demo123 (name,email) values(?,?)",
42.       [name, email],
43.       function (err, result) {
44.         if (err) {
45.           console.log(err);
46.         } else {
47.           // res.json({ result });
48.           res.redirect("/data");
49.         }
50.       }
51.     );
52.   } catch (err) {

```



```

53.     res.send(err);
54.   }
55. });
56.
57. app.listen(process.env.PORT || 3000, function (err) {
58.   if (err) console.log(err);
59.   console.log(`listening to port ${process.env.PORT}`);
60. });

```

Step 8 – Delete Operation

Let's first add a delete link to our delete button on UI

read.ejs (updated):

```

1. <!DOCTYPE html>
2. <html lang="en">
3.   <head>
4.     <meta charset="UTF-8" />
5.     <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6.     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7.     <link
8.       href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0/dist/css/bootstrap.min.css"
9.       rel="stylesheet"
10.      integrity="sha384-gH2yIJqKdNHPEq0n4Mqa/HGKIhSkIHeL5AyhkYV8i59U5AR6csBvApHHNI/vIBx"
11.      crossorigin="anonymous"
12.    />
13.
14.    <link rel="stylesheet" type="text/css" href="style.css" />
15.    <title>Document</title>
16.  </head>
17.  <body>
18.    <div class="container">
19.      <h1 class="h3 text-center">All data</h1>
20.      <div class="row">
21.        <% if(rows.length > 0) { %> <% rows.map((eachData) => { %>
22.
23.          <div class="col-md-3 col-sm-12 mx-1">
24.            <div class="card" style="width: 18rem">
25.              <div class="card-body">
26.                <h5 class="card-title"><%- eachData.name %></h5>
27.                <h6 class="card-subtitle mb-2 text-muted">
28.                  <%- eachData.email %>
29.                </h6>
30.                <a href="#" class="card-link">Edit</a>
31.                <a href="/delete-data?id=<%- eachData.ID %>" class="card-link" //delete link added
32.                  >Delete</a>
33.                >
34.              </div>
35.            </div>
36.          </div>
37.
38.        <% }) %> <%} else { %>
39.          <div class="col-md-12">
40.            <h1 class="h2 text-center">No data found</h1>
41.          </div>
42.
43.        <% }%>
44.      </div>
45.    </div>

```



```
46.   </body>
47. </html>
```

Now lets create a router for this and remove it from data base

```
1. //delete
2. app.get("/delete-data", (req, res) => {
3.   const deleteData = "delete from demo123 where id=?";
4.   connection.query(deleteData, [req.query.id], (err, rows) => {
5.     if (err) {
6.       res.send(err);
7.     } else {
8.       res.redirect("/data");
9.     }
10.   });
11. });


```

server.js updated

```
1. const express = require("express"); //express package initiated
2. const app = express(); // express instance has been created and will be access by app variable
3. const cors = require("cors");
4. const dotenv = require("dotenv");
5. var bodyParser = require("body-parser");
6. const connection = require("./config/db.js");
7.
8. dotenv.config();
9. app.use(bodyParser.urlencoded({ extended: true }));
10. app.set("view engine", "ejs");
11. app.use(cors());
12. app.use(express.json());
13. app.use(express.static(__dirname + "/public"));
14. app.use(express.static(__dirname + "/views"));
15.
16. app.get("/", (req, res) => {
17.   res.redirect("/index1.html");
18. });
19.
20. //read
21. app.get("/data", (req, res) => {
22.   const allData = "select * from demo123";
23.   connection.query(allData, (err, rows) => {
24.     if (err) {
25.       res.send(err);
26.     } else {
27.       // res.json({ rows });
28.       res.render("read.ejs", { rows });
29.     }
30.   });
31. });
32.
33. //create
34. app.post("/create", (req, res) => {
35.   var name = req.body.name;
36.   var email = req.body.email;
37.   try {
38.     connection.query(
39.       "INSERT into demo123 (name,email) values(?,?)",
40.       [name, email],
41.       function (err, result) {
42.         if (err) {
```



```

43.     console.log(err);
44. } else {
45.     // res.json({ result });
46.     res.redirect("/data");
47. }
48. }
49. );
50. } catch (err) {
51.     res.send(err);
52. }
53. });
54.
55. //delete
56. app.get("/delete-data", (req, res) => {
57.     const deleteData = "delete from demo123 where id=?";
58.     connection.query(deleteData, [req.query.id], (err, rows) => {
59.         if (err) {
60.             res.send(err);
61.         } else {
62.             res.redirect("/data");
63.         }
64.     });
65. });
66.
67. app.listen(process.env.PORT || 3000, function (err) {
68.     if (err) console.log(err);
69.     console.log(`listening to port ${process.env.PORT}`);
70. });

```

Step 9 – Update Operation

Now, update operation is quite tricky and important, so read it nicely.

1. Adding link to the Update button

read.ejs (updated)-

```

1. <!DOCTYPE html>
2. <html lang="en">
3.   <head>
4.     <meta charset="UTF-8" />
5.     <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6.     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7.     <link
8.       href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0/dist/css/bootstrap.min.css"
9.       rel="stylesheet"
10.      integrity="sha384-gH2yIJqKdNHPEq0n4Mqa/HGKIhSkIHeL5AyhkYV8i59U5AR6csBvApHHNI/vIBx"
11.      crossorigin="anonymous"
12.     />
13.
14.     <link rel="stylesheet" type="text/css" href="style.css" />
15.   <title>Document</title>
16. </head>
17. <body>
18.   <div class="container">
19.     <h1 class="h3 text-center">All data</h1>
20.     <div class="row">
21.       <% if(rows.length > 0) { %> <% rows.map((eachData) => { %>
22.
23.         <div class="col-md-3 col-sm-12 mx-1">

```



```

24. <div class="card" style="width: 18rem">
25.   <div class="card-body">
26.     <h5 class="card-title"><%- eachData.name %></h5>
27.     <h6 class="card-subtitle mb-2 text-muted">
28.       <%- eachData.email %>
29.     </h6>
30.     <a href="/update-data?id=<%- eachData.ID %>" class="card-link" //edit button link added
31.       >Edit</a
32.     >
33.     <a href="/delete-data?id=<%- eachData.ID %>" class="card-link"
34.       >Delete</a
35.     >
36.   </div>
37. </div>
38. </div>
39.
40. <% }) %> <%} else { %>
41. <div class="col-md-12">
42.   <h1 class="h2 text-center">No data found</h1>
43. </div>
44.
45. <% }%>
46. </div>
47. </div>
48. </body>
49. </html>

```

2. Passing data to update router and redirecting to the Update the page

Once user clicks on the update button , it should redirect use to an update page, and **also the input field should be pre-filled with the data**. For this, we need to hit the router which will bring the data of that particular field and populate the UI

adding in **server.js**

```

1. //passing data to update page
2. app.get("/update-data", (req, res) => {
3.   const updateData = "select * from demo123 where id=?";
4.   connection.query(updateData, req.query.id, (err, eachRow) => {
5.     if (err) {
6.       res.send(err);
7.     } else {
8.       console.log(eachRow[0]);
9.       result = JSON.parse(JSON.stringify(eachRow[0])); //in case if it dint work
10.      res.render("edit.ejs", { data: eachRow[0] });
11.    }
12.  });
13. });

```

Edit.ejs

```

1. <!DOCTYPE html>
2. <html lang="en">
3.   <head>
4.     <meta charset="UTF-8" />
5.     <meta http-equiv="X-UA-Compatible" content="IE=edge" />
6.     <meta name="viewport" content="width=device-width, initial-scale=1.0" />
7.     <link
8.       href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0/dist/css/bootstrap.min.css"
9.       rel="stylesheet"

```



```

10.    integrity="sha384-gH2yIJqKdNHPEq0n4Mqa/HGKIhSKIHeL5AyhkYV8i59U5AR6csBvApHHNI/vIIBx"
11.    crossorigin="anonymous"
12.  />
13.  <link rel="stylesheet" href="style.css" />
14.  <title>Document</title>
15.  </head>
16.  <body>
17.  <div class="container my-2">
18.    <h1 class="h1 text-center">Update Data</h1>
19.
20.    <!-- hitting the /create router -->
21.    <form id="myForm" action="http://localhost:4000/update" method="post">
22.      <input type="hidden" value="<% =data.ID %>" name="hidden_id" />
23.      <div class="mb-3">
24.        <label class="form-label">Name</label>
25.        <input
26.          type="text"
27.          class="form-control"
28.          value="<% =data.name %>"
29.          name="name"
30.        />
31.      </div>
32.      <div class="mb-3">
33.        <label class="form-label">Email address</label>
34.        <input
35.          type="email"
36.          class="form-control"
37.          value="<% =data.email %>"
38.          name="email"
39.        />
40.      </div>
41.
42.      <button type="submit" class="btn btn-primary">Update</button>
43.
44.      <input
45.        class="btn btn-outline-dark"
46.        type="button"
47.        value="No, go back!"
48.        onclick="history.go(-1)"
49.      />
50.    </form>
51.  </div>
52.  </body>
53. </html>

```

3. Actualizar la base de datos y redirigir a los usuarios a la página principal.

agregando en server.js

```

1.  //actualización final
2.  aplicación. publicar ( "/actualizar" , ( req, res ) => {
3.    constante id_data = req. cuerpo . id_oculto ;
4.    constante nombre_datos = req. cuerpo . nombre ;
5.    const email_data = req. cuerpo . correo electrónico ;
6.
7.    consola. log ( "id..." , solicitud. cuerpo . nombre , id_data );
8.
9.    const updateQuery = "actualizar demo123 establecer nombre=? , correo electrónico=? donde id=? ";
10.
11.   conexión. consulta (
12.     consulta de actualización ,

```

```

13.      [ nombre_datos, correo_electrónico_datos, id_datos ],
14.      (errar, filas) => {
15.        si (errar) {
16.          res.enviar (errar);
17.        } demás {
18.          res.redirigir (" /datos ");
19.        }
20.      }
21.    );
22.  });

```

Conclusión

Y con esto hemos llegado al final de nuestro artículo.

Espero que les encante.

Ayuda a otros

DESCARGA GRATIS



Aloje el sitio web React Js en vivo en Internet usando github

[Anterior](#)

Las mejores ofertas de amazon y flipkart para ingenieros de software

[Próximo](#)



Himanshu Shekhar

Ey. Soy ingeniero de software con más de 4,5 años. Además de mi trabajo, creo contenido sobre codificación e ingeniería de software en general. Le pediría que siguiera mi blog y mi vídeo, que le ayudarán en su viaje en ingeniería de software. Sígueme en [Youtube](#) →

[Ver todos los artículos](#)

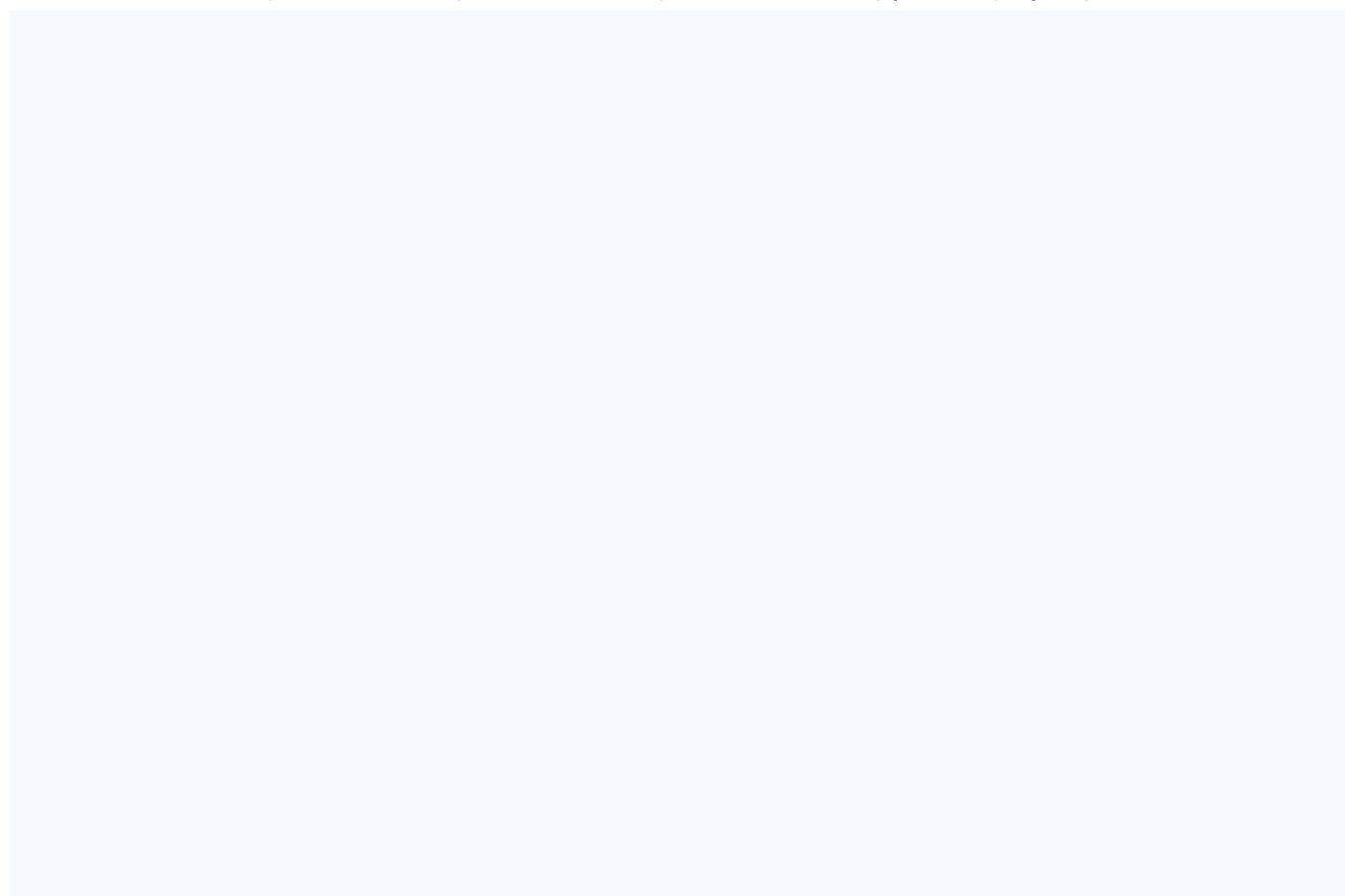
Artículos Relacionados



Cree una aplicación de pila MERN sencilla | CRUD usando

24 de febrero de 2023 Nodo JS Reaccionar Js





Aplicación CRUD Full Stack en Node JS usando ajax

28 de septiembre de 2022 Java Script Jquery Nodo JS

The diagram illustrates the transformation of a standard select box into a search-select box using Ajax. It features two side-by-side examples labeled "Before" and "After".

Before: A standard dropdown menu labeled "Select Country". The options listed are India, USA, UK, Australia, Newzeland, China, Japan, and Srilanka. The option "India" is currently selected.

After: A search-select dropdown menu labeled "Select Country". The options listed are India, USA, UK, Australia, Newzeland, China, and Japan. The option "India" is currently selected. Above the dropdown, there is a search input field with a magnifying glass icon and a placeholder "Search".

A large white arrow points from the "Before" state to the "After" state, with the text "one line of code" written vertically along its path.

www.whataaboutcoding.com

Cómo agregar una barra de búsqueda en el menú desplegable de selección

1 de abril de 2022 Javascript Jquery

3 comentarios

Aplicación CRUD Full Stack en Node JS usando ajax | ¿Qué pasa con la codificación?

[...] Escribí de septiembre de 2020 sobre mi primera aplicación CRUD usando node js y a ustedes les encantó. Pero, francamente, eso fue sólo con fines de aprendizaje, porque en la aplicación real [...]

[Responder](#)



Kunal

12 de febrero de 2023 a las 9:01 am

Muchas gracias, ahora mi comprensión básica está clara.
Ahora ayúdenme a implementar esto en mi host de cPanel.

[Responder](#)



vishal kumar

4 de mayo de 2023 a las 8:14 am

Gracias por la buena lección.

[Responder](#)

Deja una respuesta

Su dirección de correo electrónico no será publicada. Los campos obligatorios están marcados *

Comentario *

Nombre ***Correo electrónico *****Sitio web** Guardé mi nombre, correo electrónico y sitio web en este navegador para la próxima vez que comente.**publicar comentario**

Busca en este blog...

FRONTEND
Machine Coding
Round
Preparation

by **Himanshu Shekhar**

LAUNCH DATE 07 OCT SAT 23

Pre- Enroll Now

CLEAR ANY FRONTEND CODING ROUND

A large advertisement for a coding course. The main title is "FRONTEND Machine Coding Round Preparation" in white text on a dark blue background. Below it is a photo of a smiling man with a beard, pointing towards the camera with both thumbs up. To the left of the photo, the text "by Himanshu Shekhar" is written. At the bottom left, there's a button with the text "Pre- Enroll Now". On the far left, it says "LAUNCH DATE" followed by a date "07 OCT SAT 23". At the bottom right, the text "CLEAR ANY FRONTEND CODING ROUND" is displayed.

Mensajes recientes

- [Dominar la consulta del kit de herramientas de Redux: una guía completa](#)
- [Descargue la lista de más de 200 empresas de trabajos remotos](#)
- [Los 5 mejores marcos CSS para el desarrollo web en 2023](#)
- [Mutación de matriz y objeto en React Js](#)
- [Gancho personalizado en React | Guía paso por paso](#)

Plantilla del currículum vitae

Curriculum exacto que me consiguió un trabajo de 6 cifras.

Correo electrónico

[Descargar](#)

Categoría



Pregunta de la entrevista frontal	(5)
HTML/CSS	(1)
secuencia de comandos java	(11)
Serie de entrevistas sobre JavaScript	(5)
jquery	(4)
Nodo JS	(3)
Conceptos básicos de reacción	(3)
Ganchos de reacción	(3)
Reaccionar Js	(14)
Ventas y ofertas	(2)

[Política de Privacidad](#)[Mapa del sitio](#)[Términos y condiciones](#)

Copyright © 2024 ¿Qué pasa con la codificación?

