UNIÓN EUROPEA
Fondo Social Europeo
El FSE invierte en tu futuro

MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

GENERALITAT VALENCIANA
Conselleria d'Educació, Investigació, Cultura i Esport

I.E.S. SERRA PERENXISA

DWEC 2º DAW

JSON SERVER - CRUD - MODULOS - ASINC AWAIT

Curso 2.024-2025

CRM (Customer Relationship Management)

1. PREPARACIÓN PRÁCTICA

Necesitamos tener instalado node.js y vamos a instalar json.server para "crear una API"

Con node.js se instala npm (node package mode) que es un gestor de paquetes.

Debemos instalar json server a través de npm:

npm install -g json-server

Con json server simularemos la base de datos en la que vamos a hacer cambios (vamos a simular una API).

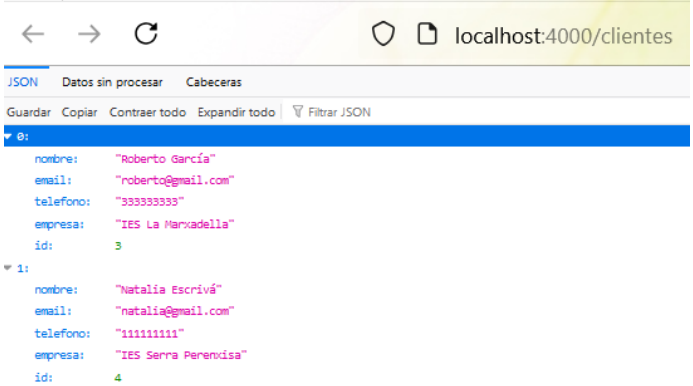
Como hemos instalado json-server de manera global, podremos acceder a él desde cualquier carpeta del sistema.

Desde la Shell nos situamos en la carpeta de la práctica y escribimos el siguiente comando para configurar nuestra API.

Tras el comando ***json-server*** indicamos el ***fichero*** y el ***puerto***. La respuesta es que carga de forma satisfactoria los datos de la ruta indicada. **Lo pararemos con CRTL+C y lo reanudaremos de la misma forma…**

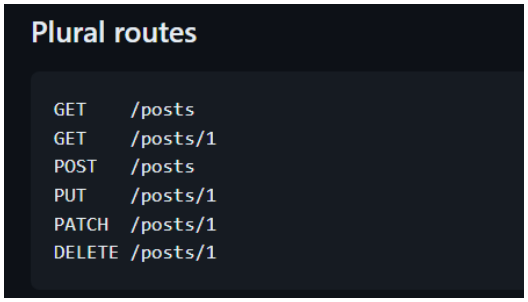
```
PS E:\CLASE\DAW\DWEC\24-25\BLOQUE2_JAVASCRIPT\UNIDAD6. MECANISMOS DE COMUNICACIÓN ASÍNCRONA. APIs\PRACTICAS\3. CRUD json server> json-server db.json -p 4000

  \{^_^}/ hi!

  Loading db.json
  Done

  Resources
  http://localhost:4000/clientes

  Home
  http://localhost:4000

  Type s + enter at any time to create a snapshot of the database
```

Con algún dato de prueba, vemos en el navegador el contenido de los datos de la API.



2. INTRODUCCIÓN RESUMEN

Partiendo de los ficheros html y css (tailwind.css), daremos vida a una aplicación web que será un CRM y en el que realizaremos las operaciones básicas con los datos (CRUD) a través de json server (https://www.npmjs.com/package/json-server)



GET: obtener datos del servidor (todos o con id)

POST: crear nuevo registro

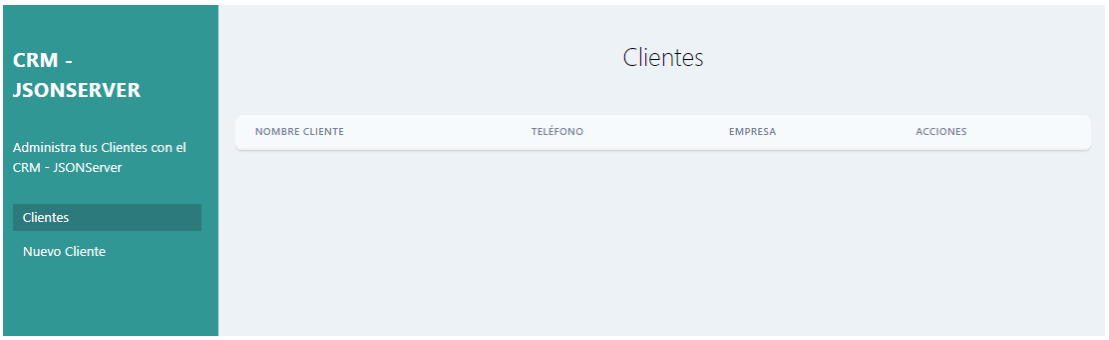
PUT: edita el objeto completo (con id) → el más usado!

PATCH: edita el objeto parcialmente (con id)

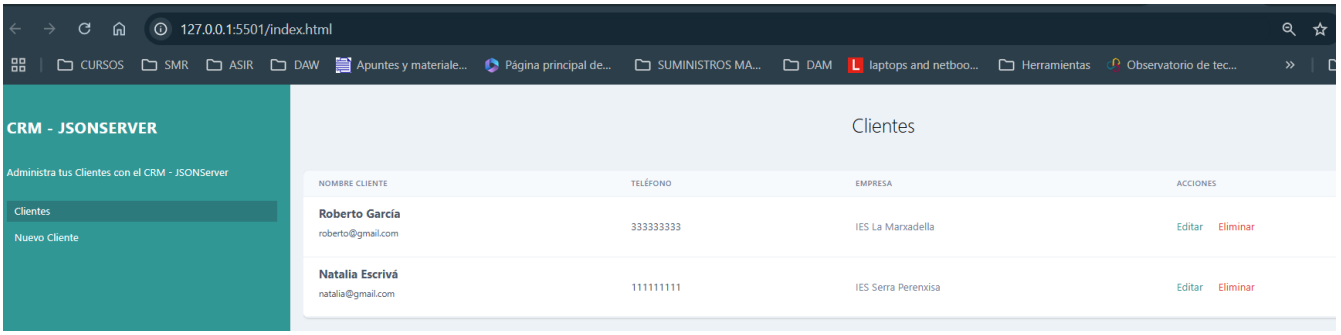
DELETE: borrar datos del servidor (con id)

Partimos de una pantalla con un menú a su izquierda en el que podemos ver los clientes y dar de alta nuevos.

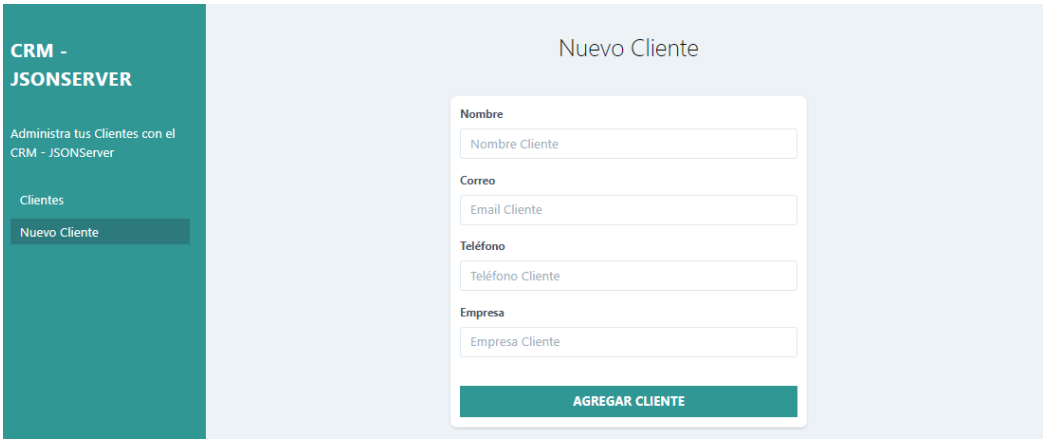
Ejemplo pantalla inicial:

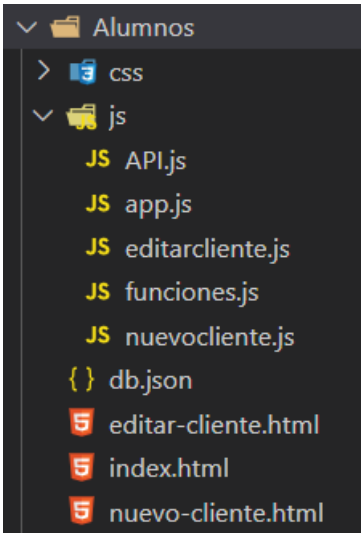


Ejemplo pantalla Clientes con datos:



Ejemplo pantalla alta cliente:





Podemos ver distintos ficheros (html y js)

En el fichero "index.html" **únicamente podremos hacer referencia a un fichero js ("app.js")**, de forma que, utilizando **módulos**, podremos hacer uso de todo el código del resto de ficheros js.

Haremos lo mismo con cada fichero html y su correspondiente fichero js.

Tendremos distintos ficheros js → Debéis organizar el código en estos ficheros de manera lógica (podéis añadir ficheros js o eliminar alguno propuesto).

3. DESARROLLO → NUEVO CLIENTE: Verificar campos formulario

Validaremos el formulario teniendo en cuenta que **todos los campos son obligatorios**.

Si el formulario no está correctamente rellenado, mostraremos un mensaje de error (estará en pantalla unos segundos únicamente). Al elemento HTML le añadiremos estas clases de tailwind:

'bg-red-100', 'border-red-400', 'text-red-700', 'px-4', 'py-3','rounded', 'max-w-lg','mx-auto', 'mt-6', 'text-center'

Si la comprobación la queremos hacer personalizada para cada apartado del formulario, usaremos una única función para el mensaje de error.

4. DESARROLLO → NUEVO CLIENTE: Añadir cliente a la BBDD

Si el formulario está bien rellenado, añadiremos a la bbdd (db.json) el nuevo cliente. Para ello usaremos async await.

Debemos hacer la llamada al servidor con fetch. La sintaxis que usábamos era fetch(url) pero, como por defecto fetch usa GET y lo que queremos es añadir un nuevo registro, usaremos **un objeto de configuración de la comunicación:**

- Method: En este caso POST
- Body: es el contenido de la petición POST al servidor. Este body se envía o como String o como Objeto. En nuestro caso se manda como String.
- Header: esto es una información de qué tipo de medio o archivo estamos enviando/solicitando. Más información https://www.iana.org/assignments/media-types/media-types.xhtml

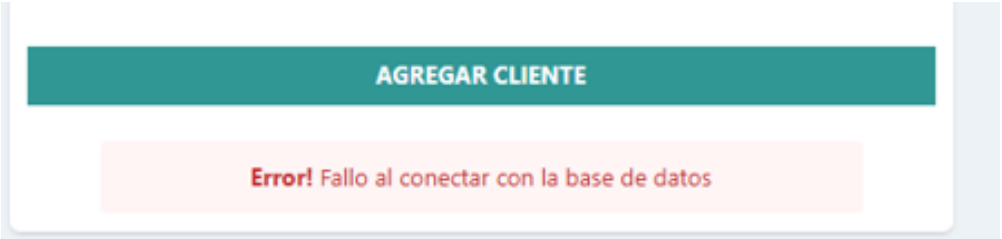
Ejemplo código fetch con parámetros de configuración (no incluye async await):

```
fetch(url,{
    method: 'POST',
    body: JSON.stringify(cliente),
    headers: {
        'Content-Type': 'application/json'
    }

})
```

Cuando hayamos dado de alta el cliente, mandaremos al usuario a la página de inicio. Para ello utilizaremos el método **location** del objeto **window** y, a su vez, la propiedad **href** a la que igualaremos a la página que queremos redirigir al usuario (***windows.location.href***)

Si vamos dando de alta clientes, vemos que nuestro fichero json se va llenando con los datos. Se puede observar que los **id autoincrementables los genera json server**.

Si, hubiese un **error**, lo mostraríamos con una alerta igual que en el punto anterior (podemos crear uno nosotros para que el usuario lo entienda)



5. DESARROLLO → MOSTRAR CLIENTES

Esta acción será la **primera que debe hacer nuestra aplicación**. Inyectaremos el HTML de cada fila a través de JS.

Las **clases de css** para los elementos son:

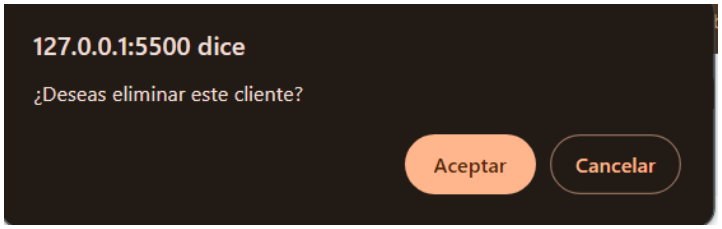
- Para **todas las celdas** (excepto las de "Acciones") son: "px-6 py-4 whitespace-no-wrap border-b border-gray-200"
- Para **nombre** : "text-sm leading-5 font-medium text-gray-700 text-lg font-bold"
- Para **email**: "text-sm leading-10 text-gray-700"
- Para **teléfono**= "text-gray-700"
- Para **empresa** = "text-gray-600"
- Añadimos además la celda con los enlaces a "Editar" y "Eliminar". Se facilita el código

```
<td class="px-6 py-4 whitespace-no-wrap border-b border-gray-200 text-sm leading-5">
   <a href="editar-cliente.html?id=${id}" class="text-teal-600 hover:text-teal-900 mr-5">Editar</a>
   <a href="#" data-cliente="${id}" class="text-red-600 hover:text-red-900 eliminar">Eliminar</a>
</td>
```

Ejemplo de salida:

6. DESARROLLO → ELIMINAR CLIENTE

Al clicar sobre el botón "Eliminar" del registro del cliente, la aplicación pedirá confirmación para realizar esta operación.

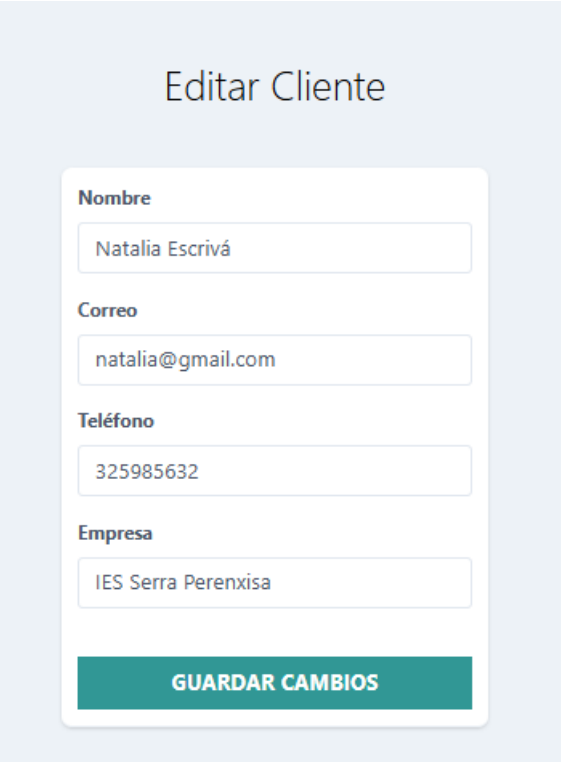


Si el usuario confirma, la aplicación deberá borrar ese cliente de la BBDD. Para ello usaremos el método 'DELETE' con el id correspondiente, tal y como se indica en las rutas de json server. No necesitamos ningún parámetro adicional como body o headers.

7. DESARROLLO → EDITAR UN CLIENTE

Al clicar sobre el botón “Editar” del registro del cliente, deberá aparecer la pantalla con los datos del cliente rellenada.

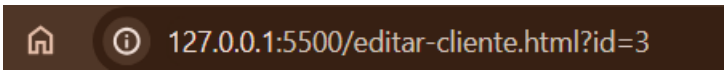
Ejemplo pantalla Editar Cliente:



Esta operación tiene dos fases:

1. Recopilar la información de la base de datos.
2. Guardar los cambios en la base de datos.

Para la **primera fase**, hasta llegar a la captura de arriba, debemos reconocer primero el cliente que está seleccionado. Cuando cliclamos sobre la opción “Editar” en la url de la página vemos algo así:



Para “recoger” ese dato lo haremos a través del **objeto de JS URLSearchParams**. Ejemplo de código:

```
const parametrosURL = new URLSearchParams(window.location.search);
const idCliente = parseInt(parametrosURL.get('id'));
```

Con estos datos, ya podremos recoger los datos del cliente (llamando a la función que conecta con la API) y mostrarlos en el formulario.

Al editar el cliente, debemos tener en cuenta que NO puede haber ningún apartado del formulario vacío. Se procederá de la misma forma que cuando damos de alta un nuevo cliente.

Para la **segunda fase**: una vez que tenemos los datos, los tenemos que guardar. Para ello, comunicaremos con la base de datos de la misma forma que hemos hecho al crear un nuevo cliente pero, en este caso usaremos como method: ‘PUT’, teniendo en cuenta que debemos actualizar solo un registro.

Llevaremos al usuario a la pantalla principal para que vea los clientes (incluyendo el actualizado).