



Fundamentos de Aplicaciones Web

LABORATORIO DE COMPUTACIÓN III

TUSI/UTN-FRA

Fundamentos de Aplicaciones Web

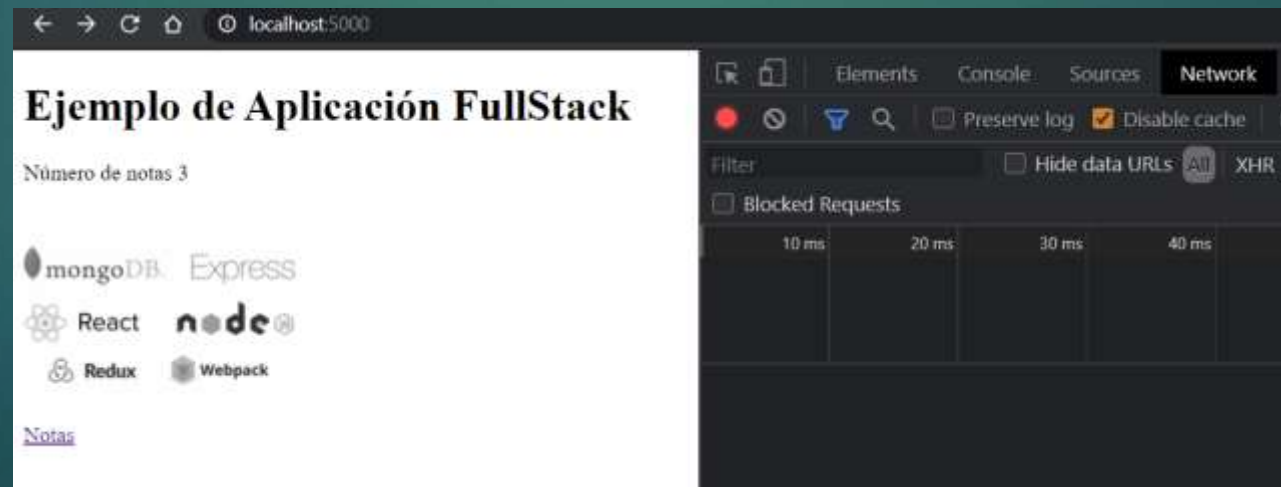
Repasaremos algunos principios del desarrollo web examinando una aplicación de ejemplo disponible en el classroom de la materia.

La aplicación existe solo para demostrar algunos conceptos básicos, y de ninguna manera son ejemplos de cómo se deben hacer las aplicaciones web. Por el contrario, demuestran algunas técnicas antiguas de desarrollo web, que incluso pueden verse como una mala práctica en la actualidad.

Se recomienda utilizar el navegador Chrome.

La primera regla del desarrollo web : mantener siempre la Consola del desarrollador abierta para eso presionamos la tecla F12.

Deshabilitamos la cache del navegador

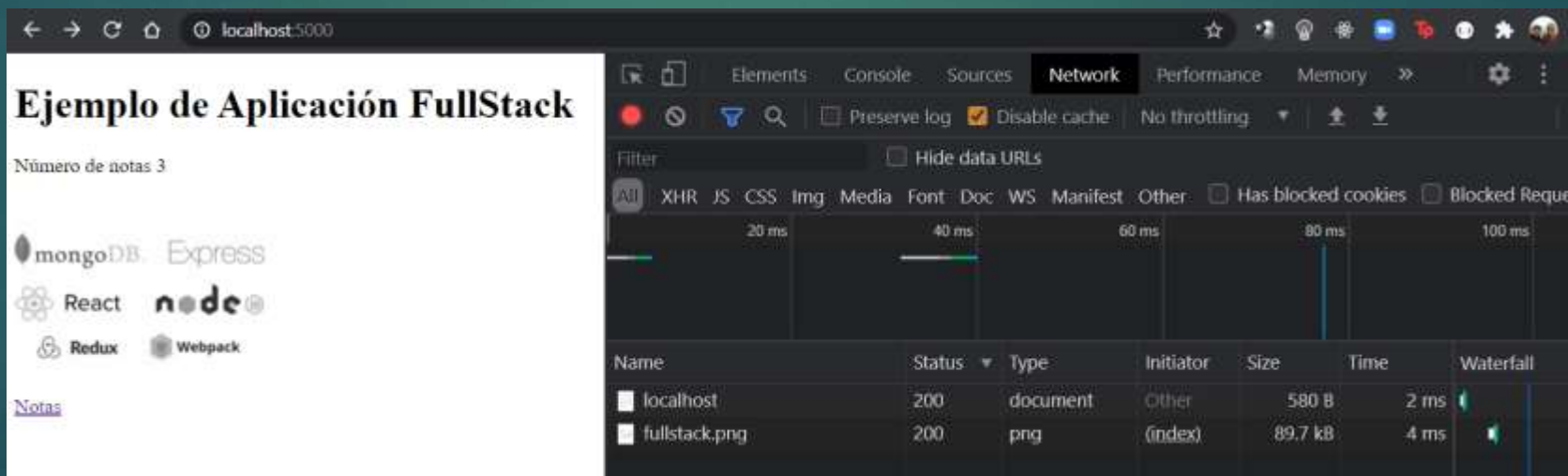


HTTP GET

El servidor y el navegador web se comunican entre sí mediante el protocolo HTTP. La pestaña Network muestra cómo se comunican el navegador y el servidor.

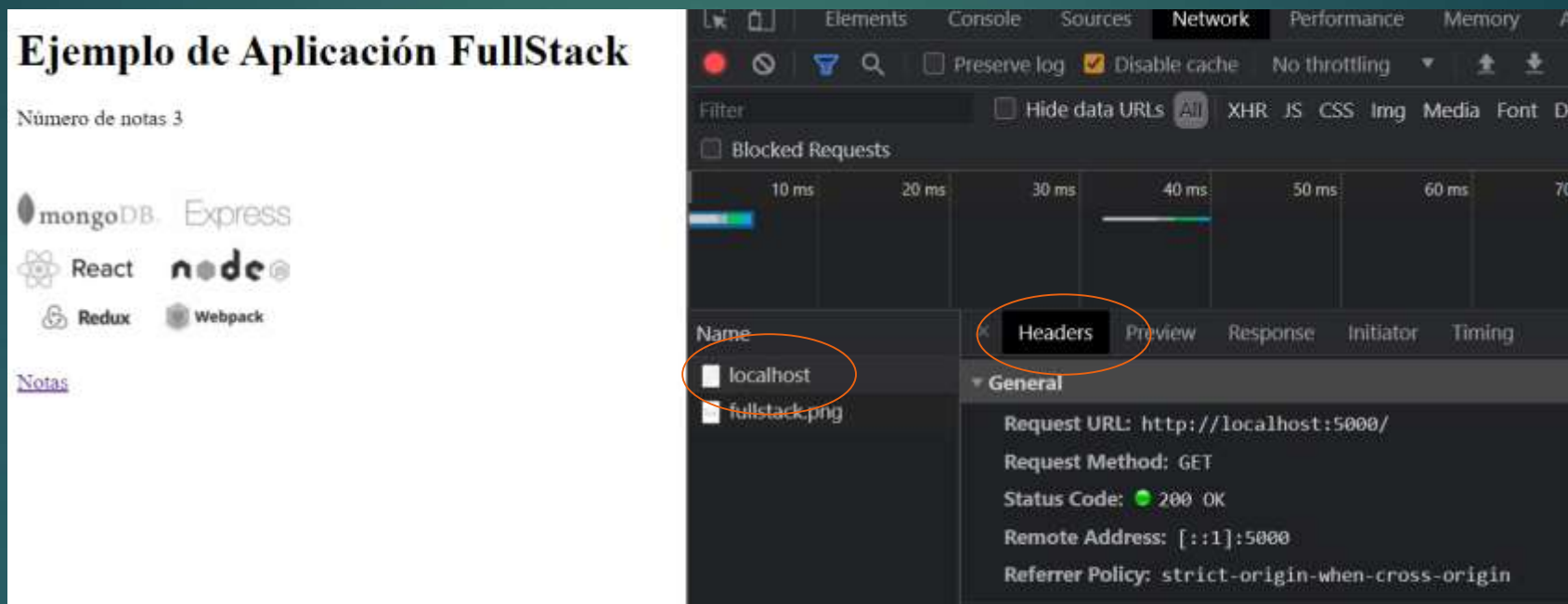
Cuando recargas la página (presionas la tecla F5 o el símbolo ↻ en el navegador), la consola muestra que han ocurrido dos eventos:

El navegador recupera el contenido de la página del localhost (servidor local de desarrollo)
Y descarga la imagen fullstack.png



HTTP GET II

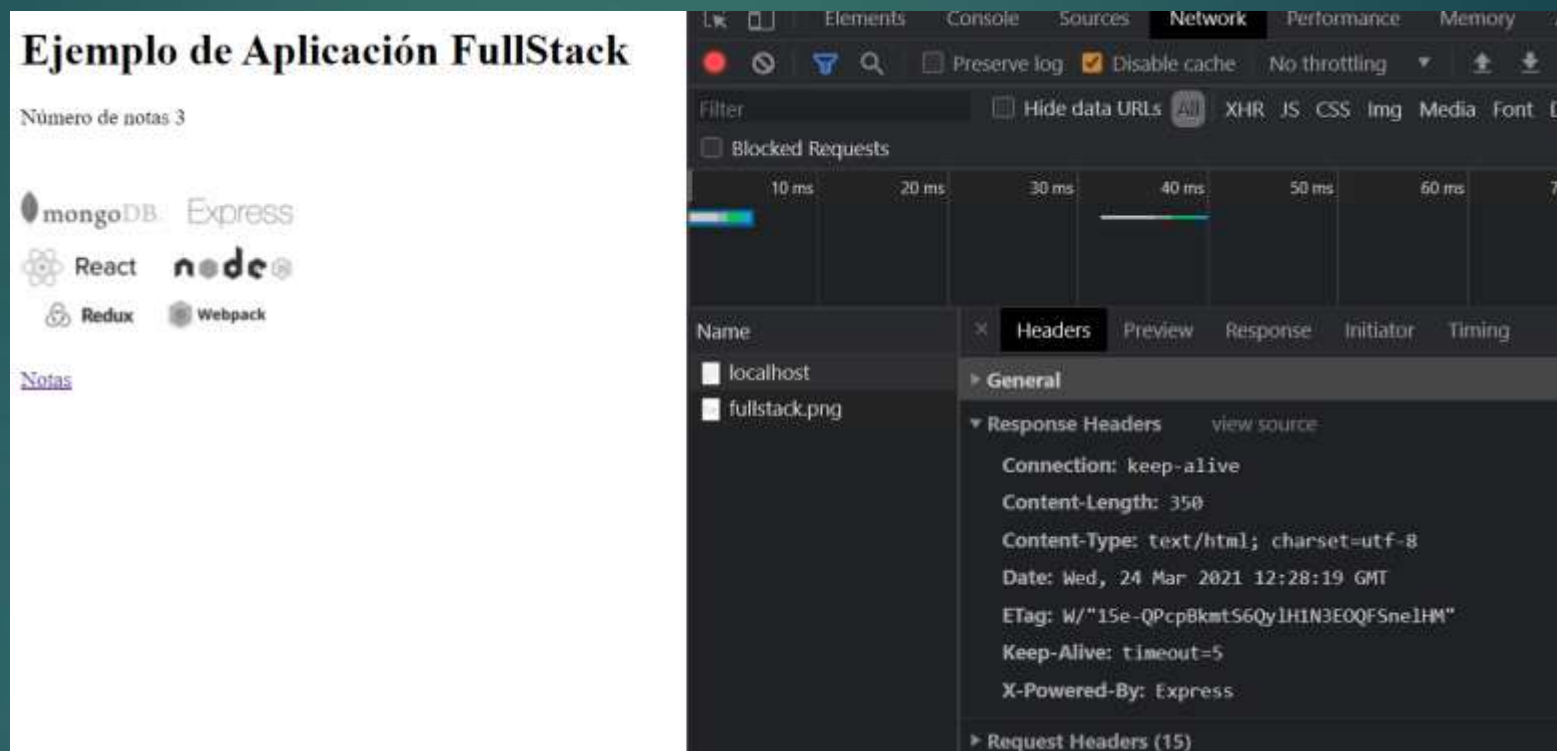
Al hacer clic en el primer evento, se muestra más información sobre lo que está sucediendo:



La pestaña General muestra que el navegador hizo una solicitud a la dirección http://localhost:5000 usando el método GET, y que la solicitud fue exitosa, porque la respuesta del servidor tiene el código de estado 200 (OK).

HTTP GET III

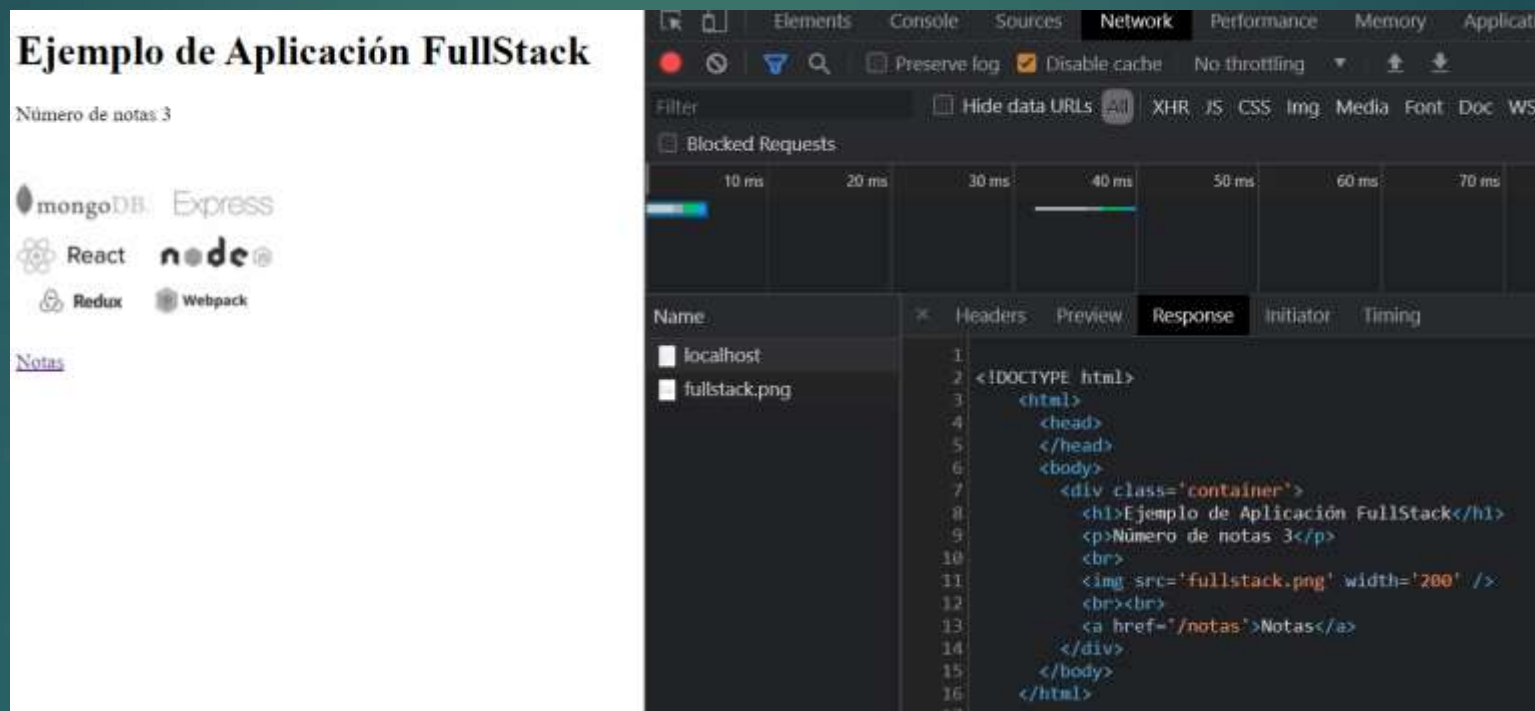
La solicitud y la respuesta del servidor tienen varias cabeceras:



Las Cabeceras de Respuesta (Response Headers) en la parte superior nos dicen, por ejemplo, el tamaño de la respuesta en bytes y la hora exacta de la respuesta. Una cabecera importante Content-Type nos dice que la respuesta es un archivo de texto en formato utf-8, cuyo contenido se ha formateado con HTML. De esta manera, el navegador sabe que la respuesta es una página HTML normal y la representa en el navegador "como una página web".

HTTP GET IV

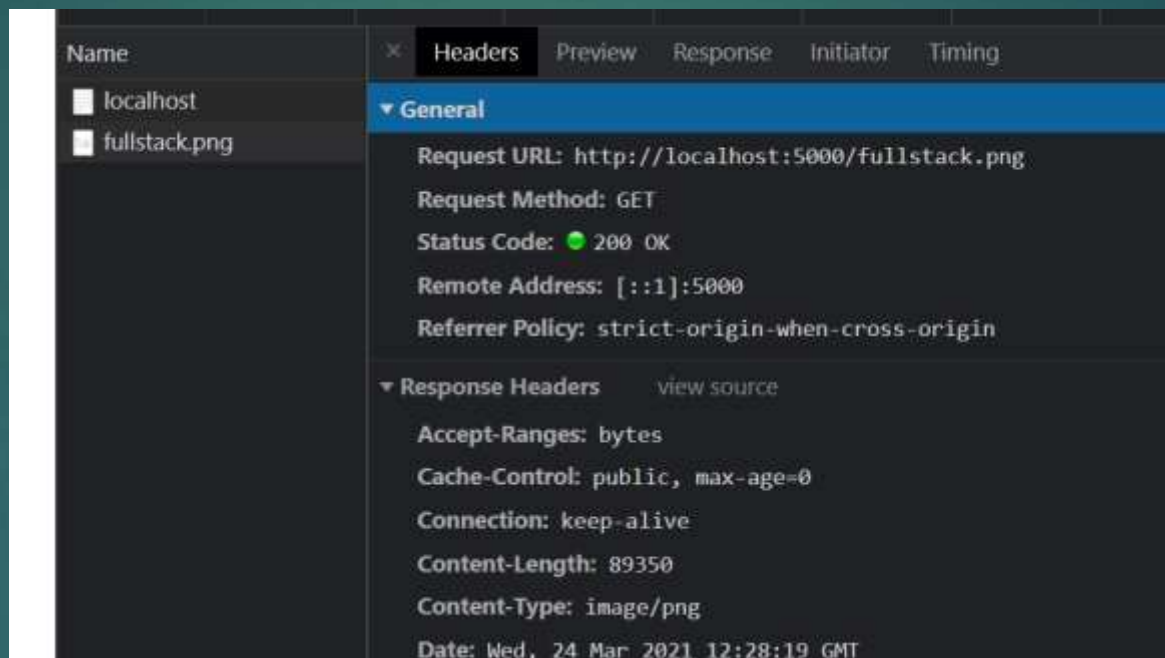
La pestaña Response muestra los datos de la respuesta, una página HTML normal. La sección body determina la estructura de la página renderizada en la pantalla:



La página contiene un elemento div, que a su vez contiene un encabezado, un enlace a la página notas y una etiqueta img, y muestra el número de notas creadas.

HTTP GET V

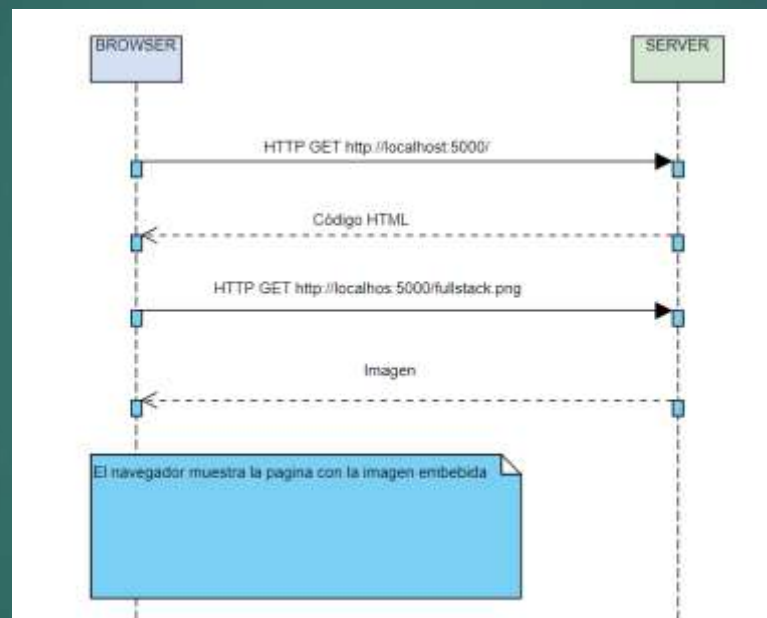
Debido a la etiqueta `img`, el navegador realiza una segunda solicitud HTTP para recuperar la imagen `fullstack.png` del servidor. Los detalles de la solicitud son los siguientes:



La solicitud se realizó a la dirección `http://localhost:5000/fullstack.png` y su tipo es HTTP GET. Las cabeceras de respuesta nos dicen que el tamaño de la respuesta es 89350 bytes y su Content-Type es `image/png`, por lo que es una imagen png. El navegador utiliza esta información para mostrar la imagen correctamente en la pantalla.

HTTP GET VI

La cadena de eventos causada por abrir la página `http://localhost:5000/` en un navegador forma el siguiente diagrama de secuencia:



Primero, el navegador realiza una solicitud HTTP GET al servidor para obtener el código HTML de la página. La etiqueta `img` en el HTML solicita al navegador que busque la imagen `fullstack.png`. El navegador muestra la página HTML y la imagen en la pantalla.

Aunque es difícil de notar, la página HTML comienza a renderizarse antes de que la imagen se haya obtenido del servidor.

Aplicaciones web tradicionales

La página de inicio de la aplicación de ejemplo funciona como una aplicación web tradicional. Al ingresar a la página, el navegador obtiene el documento HTML que detalla la estructura y el contenido textual de la página desde el servidor.

El servidor ha formado este documento de alguna manera.

El documento puede ser un archivo de texto estático guardado en el directorio del servidor, o también se pueden formar los documentos HTML dinámicamente de acuerdo con el código de la aplicación, utilizando, por ejemplo, datos de una base de datos.

El código HTML de la aplicación de ejemplo se ha formado de forma dinámica, porque contiene información sobre el número de notas creadas.

El contenido de la página HTML se guardó como un template string.

La parte que cambia dinámicamente de la página de inicio, el número de notas guardadas (en el código cantidad), se reemplaza por el número actual de notas (notas.length) en el template string.

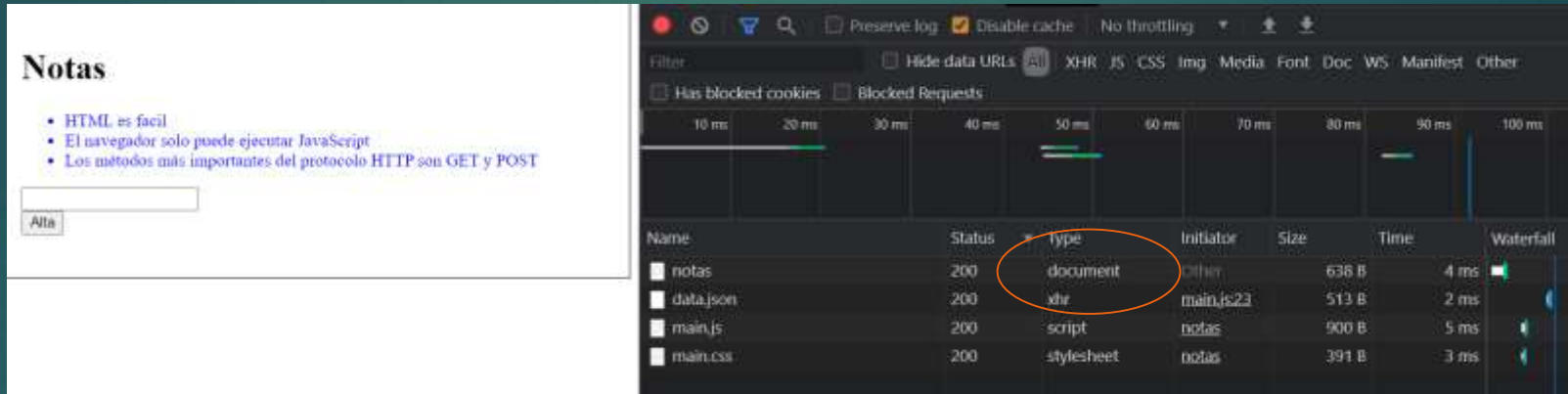
Escribir HTML en medio del código, por supuesto, no es inteligente, pero para los programadores PHP de la vieja escuela era una práctica normal.

En las aplicaciones web tradicionales, el navegador es "tonto". Solo obtiene datos HTML del servidor y toda la lógica de la aplicación está en el servidor.

```
89 const getIndexHTML = (cantidad) => {  
90   return(  
91     <!DOCTYPE html>  
92     <html>  
93       <head>  
94       </head>  
95       <body>  
96         <div class='container'>  
97           <h1>Ejemplo de Aplicación FullStack</h1>  
98           <p>Número de notas ${cantidad}</p>  
99           <br>  
100          <img src='fullstack.png' width='200' />  
101          <br><br>  
102          <a href='/notas'>Notas</a>  
103        </div>  
104      </body>  
105    </html>  
106  )  
107 }  
108
```

Ejecución de la lógica de la aplicación en el servidor

Si hacemos click en el vínculo a notas podemos ver en la pestaña Network que el navegador realiza 4 solicitudes HTTP:



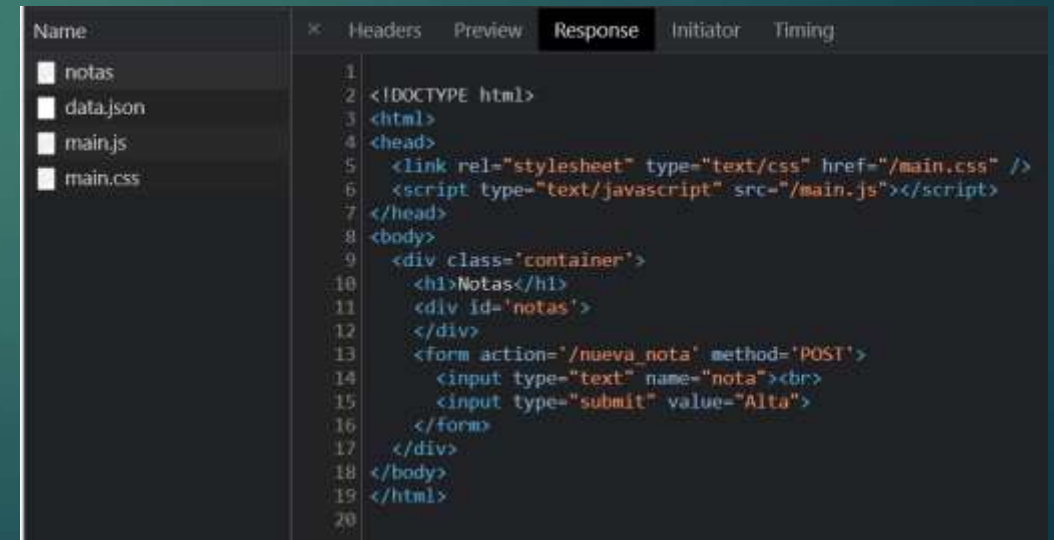
The screenshot shows a web browser with a page titled 'Notas'. The page content includes a list of bullet points: 'HTML es fácil', 'El navegador solo puede ejecutar JavaScript', and 'Los métodos más importantes del protocolo HTTP son GET y POST'. Below the text is a text input field and a button labeled 'Alta'. To the right, the Network tab is open, showing a list of four requests: 'notas', 'data.json', 'main.js', and 'main.css'. The 'notas' request is highlighted, and its details are shown in the table below.

Name	Status	Type	Initiator	Size	Time	Waterfall
notas	200	document	Other	636 B	4 ms	
data.json	200	xhr	main.js:23	513 B	2 ms	
main.js	200	script	notas	900 B	5 ms	
main.css	200	stylesheet	notas	391 B	3 ms	

Todas las solicitudes tienen tipos diferentes. El tipo de la primera solicitud es document. Es el código HTML de la página y tiene el siguiente aspecto:

Cuando comparamos la página que se muestra en el navegador y el código HTML devuelto por el servidor, notamos que el código no contiene la lista de notas.

La sección head del HTML contiene una etiqueta script, que hace que el navegador obtenga un archivo JavaScript llamado *main.js*.



The screenshot shows the Network tab with the 'notas' request selected. The 'Response' tab is active, displaying the HTML code returned by the server. The code includes a DOCTYPE declaration, a head section with a link to 'main.css' and a script for 'main.js', and a body section with a container div containing a heading 'Notas', a form with a text input and a submit button, and a footer div.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <link rel="stylesheet" type="text/css" href="/main.css" />
5   <script type="text/javascript" src="/main.js"></script>
6 </head>
7 <body>
8   <div class="container">
9     <h1>Notas</h1>
10    <div id="notas">
11    </div>
12    <form action="/nueva_nota" method="POST">
13      <input type="text" name="nota"><br>
14      <input type="submit" value="Alta">
15    </form>
16  </div>
17 </body>
18 </html>
```

El código JavaScript tiene el siguiente aspecto:

```
let xhr = new XMLHttpRequest();

xhr.onreadystatechange = function () {
  if (this.readyState == 4 && this.status == 200) {
    const data = JSON.parse(this.responseText)
    console.log(data)
    let ul = document.createElement('ul')
    ul.setAttribute('class', 'notas')

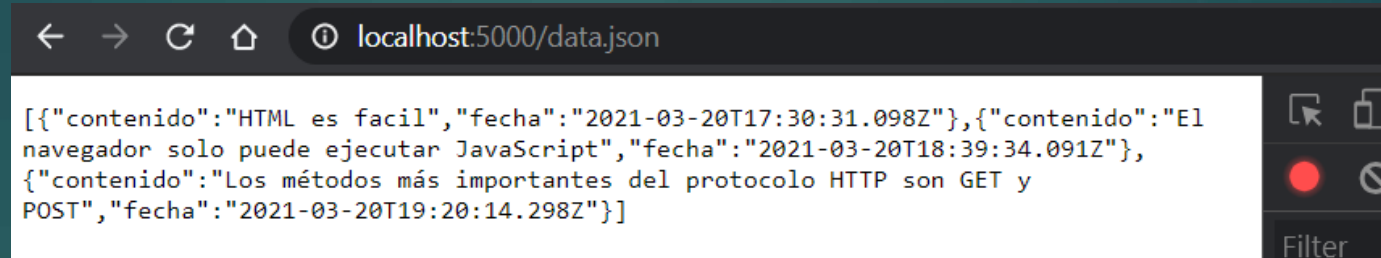
    data.forEach(function(nota){
      let li = document.createElement('li')
      ul.appendChild(li);
      li.appendChild(document.createTextNode(nota.contenido))
    })
    document.getElementById("notas").appendChild(ul)
  }
}

xhr.open("GET", "/data.json", true)
xhr.send()
```

Inmediatamente después de obtener la etiqueta script, el navegador comienza a ejecutar el código.

Las dos últimas líneas definen que el navegador realiza una solicitud HTTP GET a la dirección del servidor '/data.json'

Podemos intentar ir a la dirección `http://localhost:5000/data.json` directamente desde el navegador:

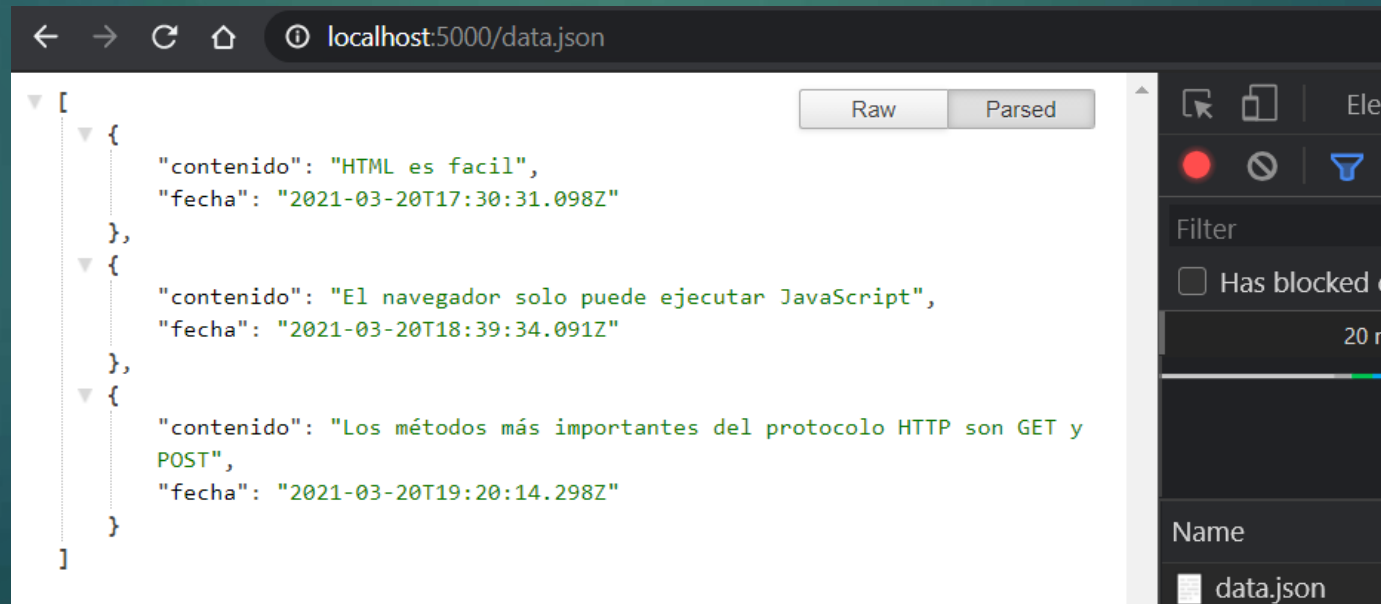


A screenshot of a web browser window. The address bar shows `localhost:5000/data.json`. The main content area displays a raw JSON array of three objects. The right sidebar contains icons for back, forward, and search, along with a 'Filter' button.

```
[{"contenido": "HTML es facil", "fecha": "2021-03-20T17:30:31.098Z"}, {"contenido": "El navegador solo puede ejecutar JavaScript", "fecha": "2021-03-20T18:39:34.091Z"}, {"contenido": "Los m\u00e9todos m\u00e1s importantes del protocolo HTTP son GET y POST", "fecha": "2021-03-20T19:20:14.298Z"}]
```

Encontramos las notas como "datos sin procesar" en JSON

De forma predeterminada, el navegador no es demasiado bueno para mostrar datos JSON. Se pueden usar complementos para manejar el formato. Instalemos JSON Formatter en Chrome y volvamos a cargar la p\u00e1gina. Los datos ahora est\u00e1n bien formateados:



A screenshot of a web browser window with the JSON Formatter extension installed. The address bar shows `localhost:5000/data.json`. The main content area displays the same JSON data as the previous screenshot, but it is now formatted with syntax highlighting and collapsible nodes. The right sidebar shows the 'Filter' button and a 'Has blocked co' checkbox. The bottom right corner shows the file name 'data.json'.

```
[
  {
    "contenido": "HTML es facil",
    "fecha": "2021-03-20T17:30:31.098Z"
  },
  {
    "contenido": "El navegador solo puede ejecutar JavaScript",
    "fecha": "2021-03-20T18:39:34.091Z"
  },
  {
    "contenido": "Los m\u00e9todos m\u00e1s importantes del protocolo HTTP son GET y POST",
    "fecha": "2021-03-20T19:20:14.298Z"
  }
]
```


Entonces, el código JavaScript de la página de notas anterior descarga los datos JSON que contienen las notas y forma una lista no ordenada a partir de este origen de datos. Esto se hace mediante el siguiente código:

```
5      const data = JSON.parse(this.responseText)
6      console.log(data)
7      let ul = document.createElement('ul')
8      ul.setAttribute('class', 'notas')
9
10     data.forEach(function(nota){
11         let li = document.createElement('li')
12         ul.appendChild(li);
13         li.appendChild(document.createTextNode(nota.contenido))
14     })
15     document.getElementById("notas").appendChild(ul)
```

En la línea 5 se convierte el json (texto) en un objeto de javascript (array de notas), en la línea 7 se crea la lista no ordenada y en la 10 se recorre el array y se crea un list item por cada nota donde el contenido del li es el contenido de la nota. En este ejemplo no utilizamos el campo fecha para nada.

La línea 15 agregar a la pagina la lista creada dinámicamente.

Event handlers y Callbacks

La estructura de este código es un poco extraña

```
1  let xhr = new XMLHttpRequest();
2
3  xhr.onreadystatechange = function () {
4    // Código que se encarga de la respuesta del servidor
5  }
6  xhr.open("GET", "/data.json", true)
7  xhr.send()
8
```

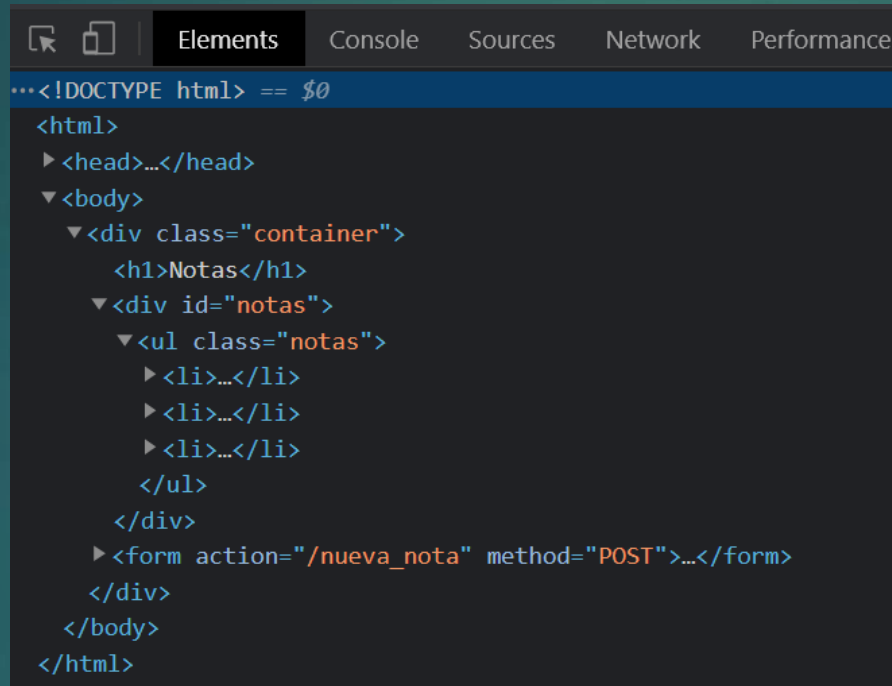
La solicitud al servidor se envía en la última línea, pero el código para manejar la respuesta se especifica a partir de la línea 3. ¿Qué está pasando?

Se define un controlador de eventos para el evento **onreadystatechange** del objeto **xhr** que realiza la solicitud. Cuando cambia el estado del objeto, el navegador llama a la función del controlador de eventos. El código de la función verifica que `readyState` sea igual a 4 (que describe la situación La operación está completa) y que el código de estado HTTP de la respuesta es 200.

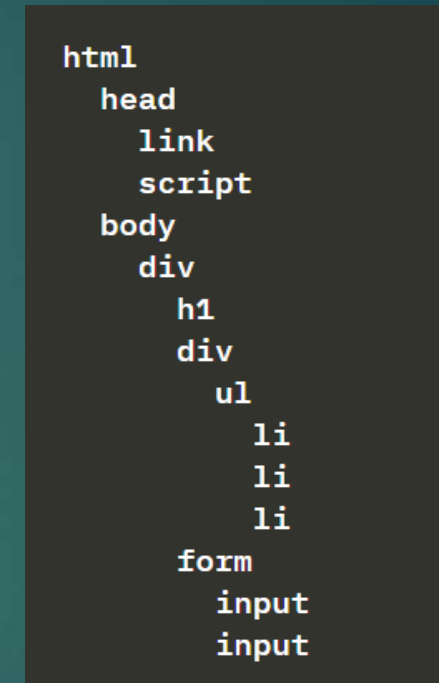
El mecanismo de invocación de controladores de eventos es muy común en JavaScript. Las funciones del controlador de eventos se denominan funciones callback. El código de la aplicación no invoca las funciones en sí, sino el entorno de ejecución -el navegador-, invoca la función en el momento adecuado, cuando se ha producido el evento.

Modelo de Objeto de Documento o DOM

Podemos pensar en las páginas HTML como estructuras de árbol implícitas. La misma estructura de árbol se puede ver en la pestaña Elements.



```
...<!DOCTYPE html> == $0
<html>
  <head>...</head>
  <body>
    <div class="container">
      <h1>Notas</h1>
      <div id="notas">
        <ul class="notas">
          <li>...</li>
          <li>...</li>
          <li>...</li>
        </ul>
      </div>
      <form action="/nueva_nota" method="POST">...</form>
    </div>
  </body>
</html>
```



```
html
  head
    link
    script
  body
    div
      h1
      div
        ul
          li
          li
          li
      form
        input
        input
```

El funcionamiento del navegador se basa en la idea de representar los elementos HTML como un árbol.

Document Object Model, o DOM es una interfaz de programación de aplicaciones, (una API), que permite la modificación programática de árboles de elementos correspondientes a páginas web.

CSS

El elemento head del código HTML de la página de Notes contiene un enlace, que determina que el navegador debe obtener una hoja de estilo CSS de la dirección main.css.


Las hojas de estilo en cascada (CSS) es un lenguaje de marcado que se utiliza para determinar la apariencia de las páginas web.

El archivo CSS obtenido tiene el siguiente aspecto:

El archivo define dos selectores de clase. Se utilizan para seleccionar ciertas partes de la página y definir reglas de estilo para aplicarles.

Una definición de selector de clase siempre comienza con un punto y contiene el nombre de la clase.

Las clases son atributos, que se pueden agregar a elementos HTML.

```
public >  main.css > ...  
1   .container {  
2   |   padding: 40px;  
3   |   border: 1px solid  
4   |  
5   .notas {  
6   |   color:  blue;  
7   |  
7   }
```

CSS II

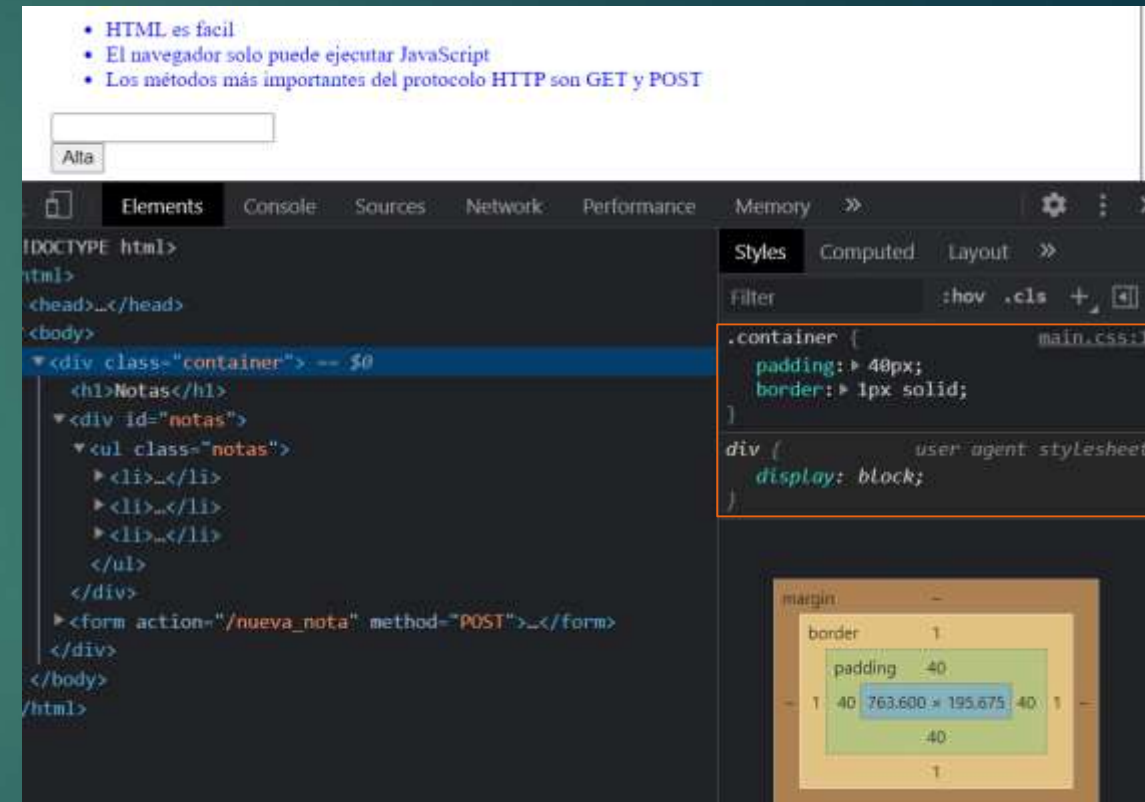
El elemento div más externo tiene la clase container. El elemento ul que contiene la lista de notas tiene la clase notes.

La regla CSS define que los elementos con la clase container se delinearán con un border de un píxel de ancho. También establece un padding de 10 píxeles en el elemento. Esto agrega un espacio vacío entre el contenido del elemento y el borde.

La segunda regla CSS establece el color del texto de las notas en azul.

Los elementos HTML también pueden tener otros atributos además de clases. El elemento div que contiene las notas tiene un atributo id. El código JavaScript usa el id para encontrar el elemento.

La pestaña Elements de la consola se puede utilizar para cambiar los estilos de los elementos.



Cargando una página que contiene JavaScript

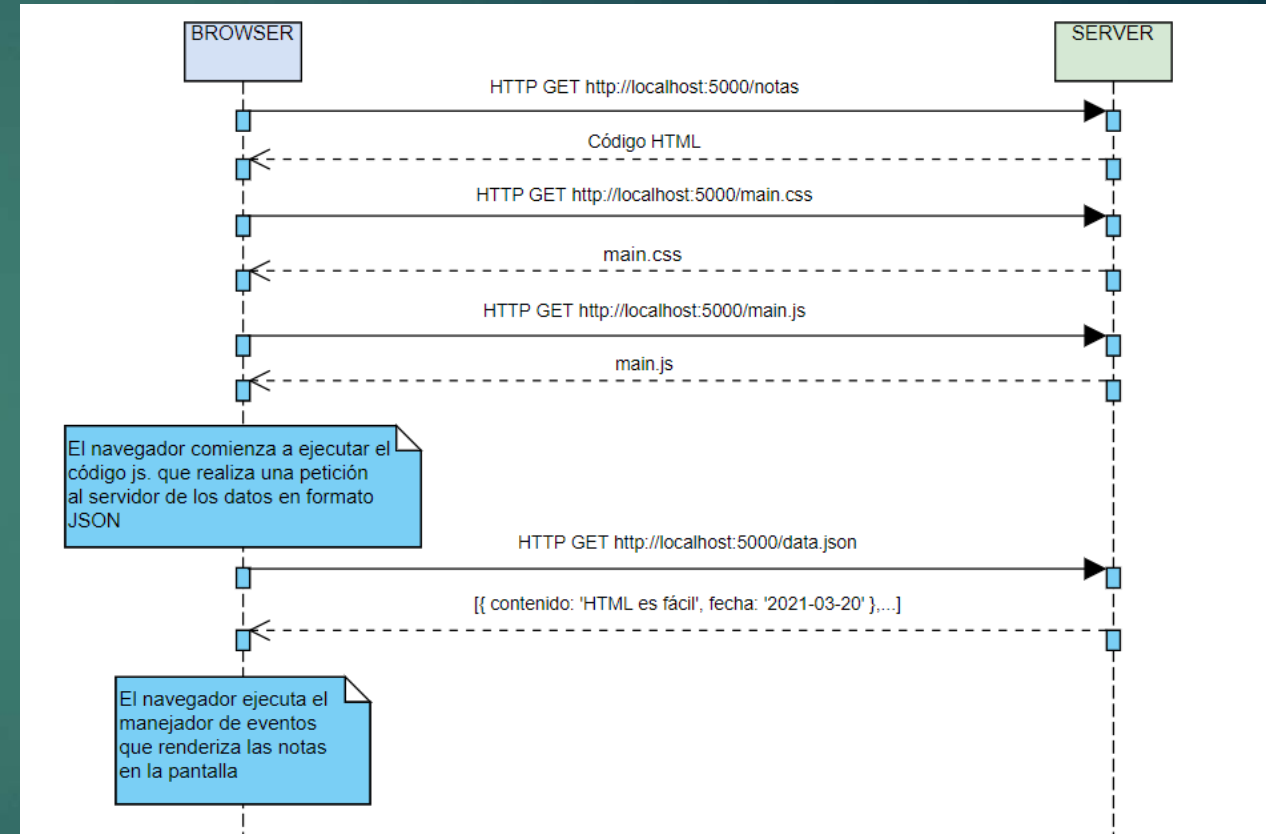
El navegador obtiene el código HTML que define el contenido y la estructura de la página del servidor mediante una solicitud HTTP GET.

Los enlaces en el código HTML hacen que el navegador también busque la hoja de estilo CSS main.cs...

... y un archivo de código JavaScript main.js

El navegador ejecuta el código JavaScript. El código realiza una solicitud HTTP GET a la dirección `https://localhost:5000/data.json`, que devuelve las notas como datos JSON.

Cuando se han obtenido los datos, el navegador ejecuta un controlador de eventos, que muestra las notas en la página utilizando DOM-API.



Formularios y HTTP POST

A continuación, examinemos cómo se realiza la adición de una nueva nota.
La página de notas contiene un elemento de formulario

Notas

- HTML es facil
- El navegador solo puede ejecutar JavaScript
- Los métodos más importantes del protocolo HTTP son GET y POST

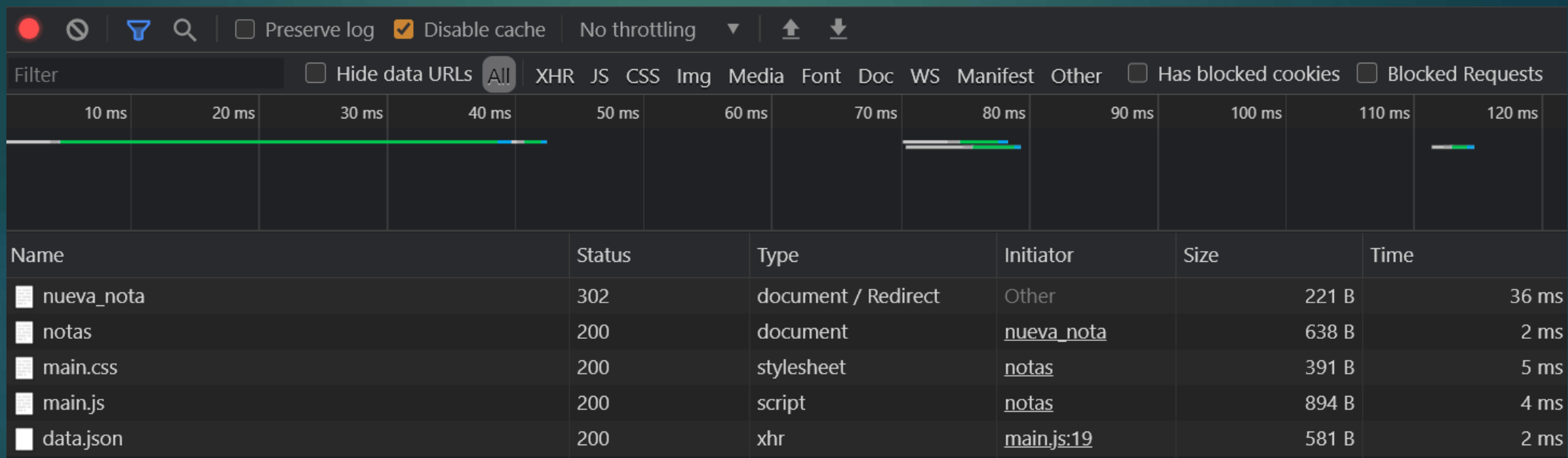
```
</div>
<form action="/nueva_nota" method="POST">
  <input type="text" name="nota">
  <br>
  <input type="submit" value="Alta">
</form>
</div>
</body>
</html>
```

html

Cuando se hace clic en el botón del formulario, el navegador enviará la entrada del usuario al servidor.

Envío por POST

Abramos la pestaña Network y veamos cómo se ve enviar el formulario:



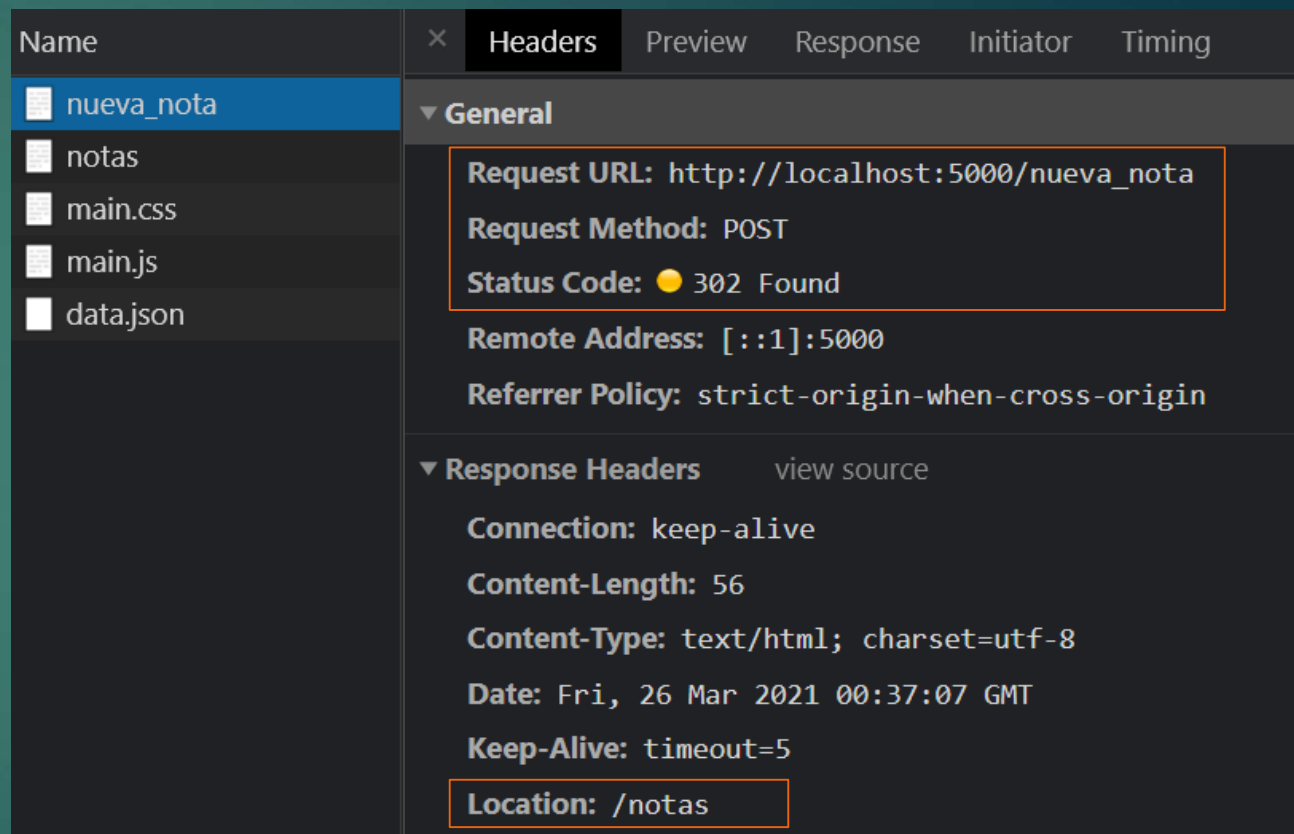
Sorprendentemente, enviar el formulario causa en total cinco solicitudes HTTP. El primero es el evento de envío de formulario.

Envío por POST II

Es una solicitud HTTP POST a la dirección del servidor /nueva_nota. El servidor responde con el código de estado HTTP 302. Se trata de una redirección con la que el servidor solicita al navegador que realice una nueva solicitud HTTP GET a la dirección definida en la Ubicación (Location) del encabezado: la dirección notas.

Entonces, el navegador vuelve a cargar la página de Notas.

La recarga provoca tres solicitudes HTTP más: obtener la hoja de estilo (main.css), el código JavaScript (main.js) y los datos sin procesar de las notas (data.json).



The screenshot displays the 'Headers' tab of a web browser's developer tools. On the left, a list of resources is shown: 'nueva_nota' (selected), 'notas', 'main.css', 'main.js', and 'data.json'. The main panel shows the details for the selected 'nueva_nota' request.

General

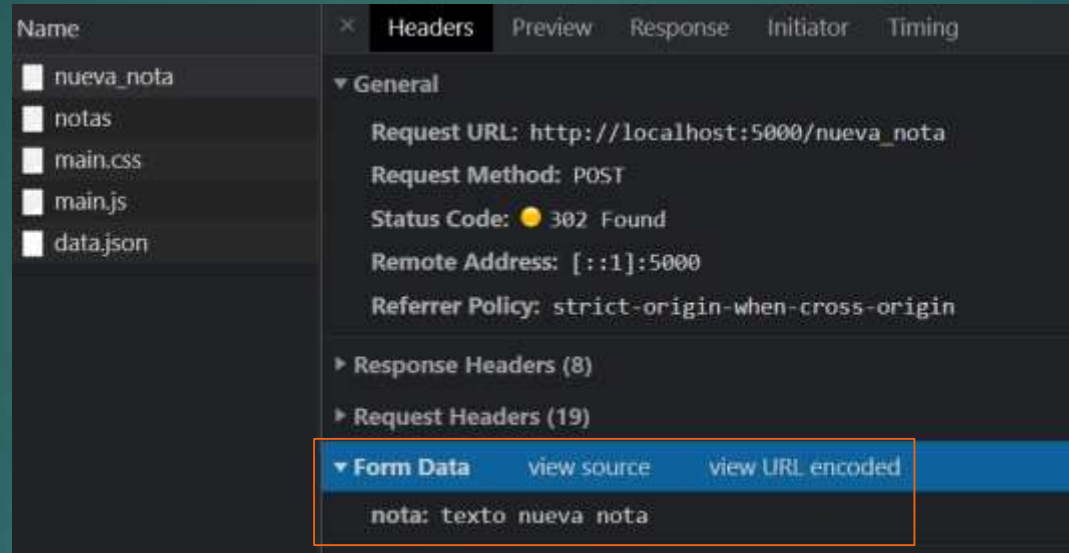
- Request URL:** http://localhost:5000/nueva_nota
- Request Method:** POST
- Status Code:** 302 Found
- Remote Address:** [::1]:5000
- Referrer Policy:** strict-origin-when-cross-origin

Response Headers [view source](#)

- Connection:** keep-alive
- Content-Length:** 56
- Content-Type:** text/html; charset=utf-8
- Date:** Fri, 26 Mar 2021 00:37:07 GMT
- Keep-Alive:** timeout=5
- Location:** /notas

Envío por POST III

La pestaña network también muestra los datos enviados con el formulario:



La etiqueta Form tiene atributos action y method, que definen que el envío del formulario se realiza como una solicitud HTTP POST a la dirección /nueva_nota

```
><div id="notas">...</div>
><form action="/nueva_nota" method="POST">
  <input type="text" name="nota">
  <br>
  <input type="submit" value="Alta">
</form>
</div>
```

Envío por POST IV

El código en el servidor responsable de la solicitud POST es bastante simple (Aclaración: este código está en el servidor, y no en el código JavaScript obtenido por el browser):

```
router.post('/nueva_nota', (req, res) => {  
  notas.push({  
    contenido: req.body.nota,  
    fecha: new Date()  
  })  
  res.redirect('/notas')  
})
```

Los datos se envían en el cuerpo de la solicitud POST.

El servidor puede acceder a los datos accediendo al campo req.body del objeto de solicitud req.

El servidor crea un nuevo objeto nota y lo agrega a un array llamado notas.

Los objetos Nota tienen dos campos: contenido que contiene el contenido real de la nota y fecha que contiene la fecha y hora en que se creó la nota.

El servidor no guarda nuevas notas en una base de datos, por lo que las nuevas notas desaparecen cuando se reinicia el servidor.

AJAX

La página Notas de la aplicación sigue un estilo de desarrollo web de los noventa y "utiliza Ajax". Como tal, está en la cresta de la ola de tecnología web de principios de la década de 2000.

AJAX (JavaScript Asincrónico y XML) es un término introducido en febrero de 2005 sobre la base de los avances en la tecnología de los navegadores para describir un nuevo enfoque revolucionario que permitió la obtención de contenido en páginas web utilizando JavaScript incluido dentro del HTML, sin la necesidad de volver a renderizar la página.

Antes de la era AJAX, todas las páginas web funcionaban como la aplicación web tradicional que vimos anteriormente. Todos los datos que se muestran en la página se obtuvieron con el código HTML generado por el servidor.

La página Notas utiliza AJAX para obtener los datos de las notas. El envío del formulario todavía utiliza el mecanismo tradicional de envío de formularios web.

Las URLs de la aplicación reflejan las viejas prácticas. Los datos JSON se obtienen de la URL `http://localhost:5000/data.json` y se envían nuevas notas a la URL `http://localhost:5000/nueva_nota`.

Hoy en día, URLs como estas no se considerarían aceptables, ya que no siguen las convenciones generalmente reconocidas de las API RESTful


SPA - Single Page Application

Vamos a ejecutar una versión de la aplicación que implementa una spa. Para eso usemos la URL <http://localhost:5000/spa>. A primera vista, la aplicación se ve exactamente igual que la anterior. El código HTML es casi idéntico, pero el archivo JavaScript es diferente (spa.js) y hay un pequeño cambio en cómo se define la etiqueta de formulario:

El formulario no tiene atributos de action o method para definir cómo y dónde enviar los datos de entrada.



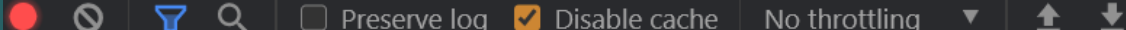
SPA

Abrimos la pestaña Network y la limpiamos haciendo clic en el símbolo . Creamos una nueva nota y observamos que el navegador envía solo una solicitud al servidor.


Notas -- SPA

- HTML es facil
- El navegador solo puede ejecutar JavaScript
- Los métodos más importantes del protocolo HTTP son GET y POST
- Las SPA no recargan la página

ElementsConsoleSourcesNetworkPerformanceMemoryApplicationSecurityLighthouse



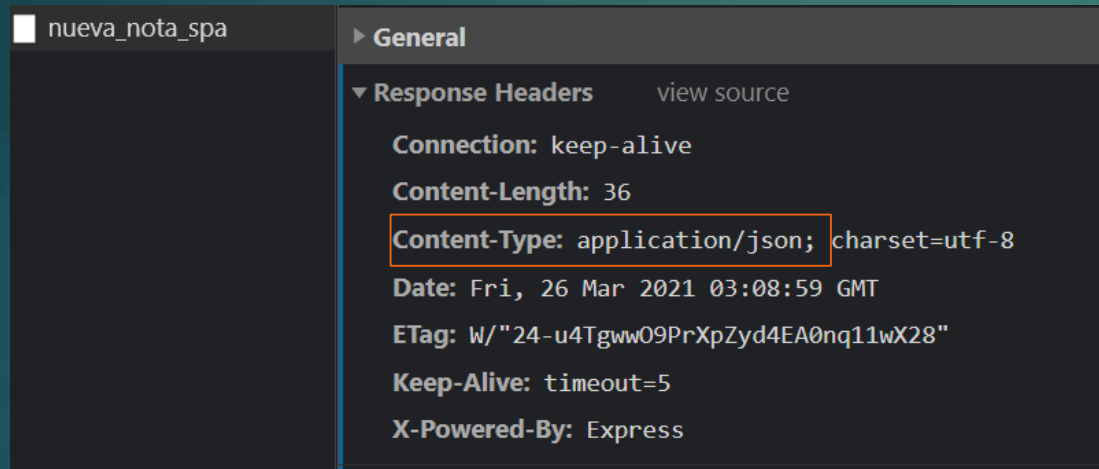
Filter☐ Hide data URLsAllXHRJSCSSImgMediaFontDocWSManifestOther☐ Has

Name	Status	Type	Initiator
 nueva_notas SPA	201	xhr	spa.js:42

SPA

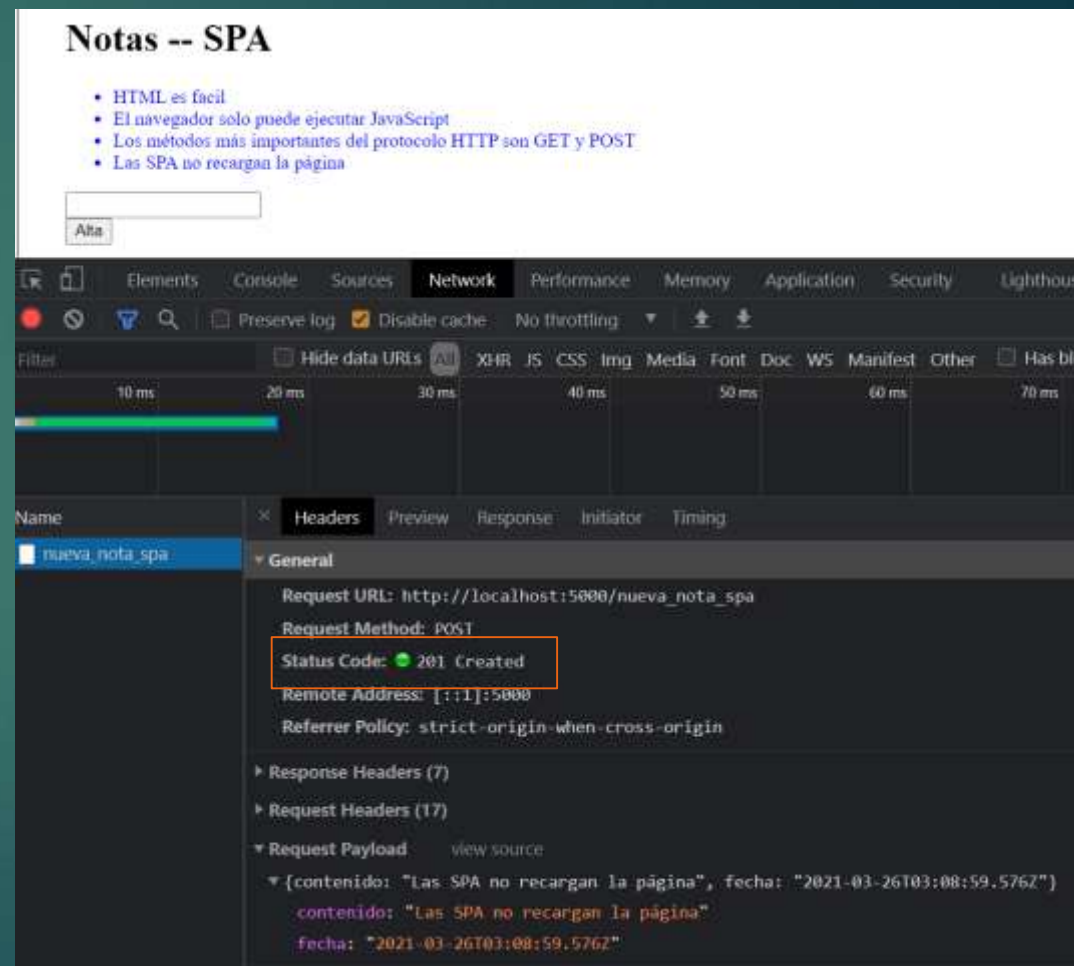
La solicitud POST a la dirección nueva_notas_spa contiene la nueva nota como datos JSON que contienen tanto el contenido de la nota (contenido) como la marca de tiempo (fecha):

La cabecera Content-Type de la solicitud le dice al servidor que los datos incluidos están representados en formato JSON.



Sin esta cabecera, el servidor no sabría cómo analizar correctamente los datos.

El servidor responde con el código de estado 201 Created. Esta vez, el servidor no solicita una redirección, el navegador permanece en la misma página y no envía más solicitudes HTTP.



SPA

La versión SPA de la aplicación no envía los datos del formulario de la forma tradicional, sino que utiliza el código JavaScript que obtuvo del servidor.

El comando `document.getElementById('frm_notas')` le indica al código que busque el elemento form de la página para registrar un manejador de eventos para el evento de envío del formulario.

El manejador llama inmediatamente al método `e.preventDefault()` para evitar el comportamiento predeterminado del envío de formularios.

El método predeterminado enviaría los datos al servidor y provocaría una nueva solicitud GET, lo que no queremos que suceda.

Después el manejador de eventos crea una nueva nota, la agrega al array de notas con la sentencia `notas.push(nota)`, actualiza la lista de notas en la página y envía la nueva nota al servidor.

```
let form = document.getElementById("frm_notas");
form.onsubmit = function (e) {
    e.preventDefault();

    let nota = {
        contenido: e.target.elements[0].value,
        fecha: new Date(),
    };

    notas.push(nota);
    e.target.elements[0].value = "";
    actualizarNotas();
    altaNota(nota);
};
```


SPA

El código para enviar la nota al servidor es el siguiente:

```
let altaNota = function (nota) {  
  let xhrPost = new XMLHttpRequest();  
  xhrPost.onreadystatechange = function () {  
    if (this.readyState == 4 && this.status == 201) {  
      console.log(this.responseText);  
    }  
  };  
  
  xhrPost.open("POST", "/nueva_nota_spa", true);  
  xhrPost.setRequestHeader("Content-type", "application/json");  
  xhrPost.send(JSON.stringify(nota));  
};
```

El código determina que los datos se enviarán con una solicitud HTTP POST y el tipo de datos será JSON. El tipo de datos se determina con una cabecera Content-type. Luego, los datos se envían como JSON-string.