



Bootcamp: Full Stack Open. Parte 0.

Con apoyo del curso de Midudev.

Aprender a usar GitHub:

<https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners>

Instalar Node.js:

<https://nodejs.org/en>

Repositorio del proyecto:

<https://github.com/fullstack-hy2020/fullstack-hy2020.github.io/tree/source>

Node.js.

- Es orientado a eventos y asíncrono.
- Contiene su propio gestor de paquetes, llamado *npm*.

HTTP Web: Consola del desarrollador.

Network (red).

- Disable cache (Deshabilitar de caché).
- Preserve log (Preservar registros).



Eventos.

Eventos.

En Headers.

General.

1. Dirección.
2. Método:
 - a. GET: <https://developer.mozilla.org/es/docs/Web/HTTP/Methods/GET>.
3. Estado:
 - a. 200 (OK): https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos_de_estado_HTTP

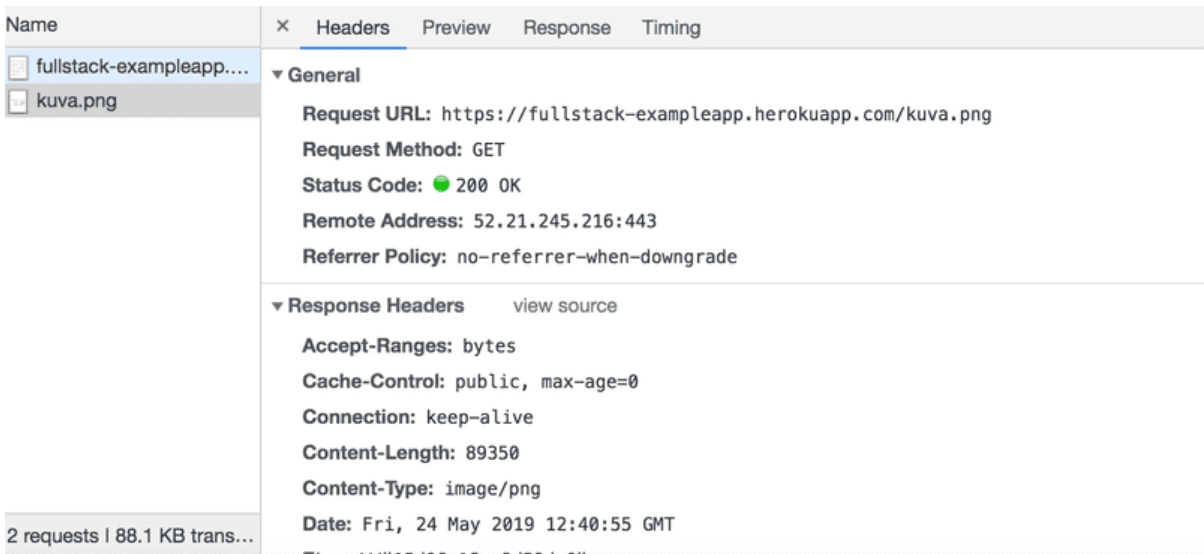
Response Headers.

1. Tamaño de la respuesta en bytes.
2. Hora exacta de la respuesta.
3. Content type.
 - a. Tipo de archivo.
 - b. Formato (utf-8): <https://es.wikipedia.org/wiki/UTF-8>

En Response.

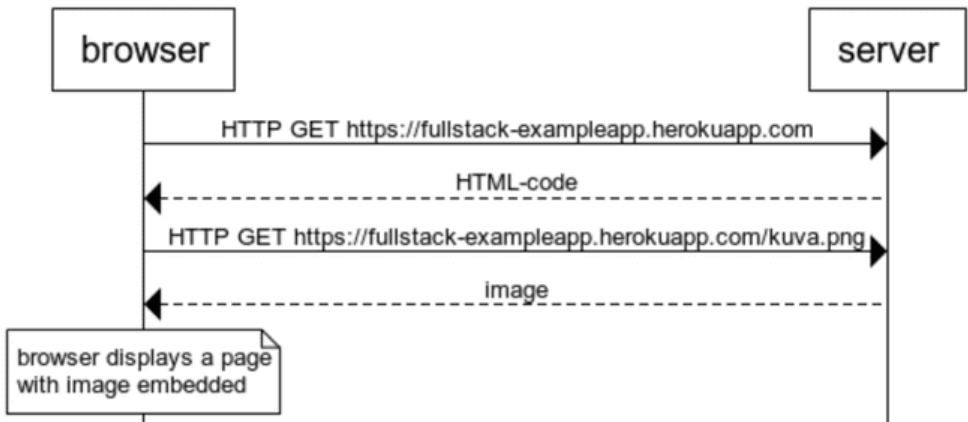
- 1. Muestra el HTML.
 - a. body.

Resumen.



La solicitud se realizó a la dirección <https://studies.cs.helsinki.fi/exampleapp/kuva.png> y su tipo es HTTP GET. Las Response Headers 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.

La cadena de eventos causada por abrir la página <https://studies.cs.helsinki.fi/exampleapp> en un navegador forma el siguiente diagrama de secuencia:



El diagrama de secuencia visualiza cómo el navegador y el servidor se comunican a lo largo del tiempo. El tiempo fluye en el diagrama de arriba a abajo, por lo que el diagrama comienza con la primera solicitud que el navegador envía al servidor, seguida de la respuesta.

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 *kuva.png*. El navegador muestra la página HTML y la imagen en la pantalla.

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

Aplicaciones web tradicionales.



En las aplicaciones web tradicionales, el navegador es "tonto". Solo obtiene datos HTML del servidor, y toda la lógica de la aplicación reside en el servidor. Un servidor puede ser creado utilizando Java Spring, Python Flask, o Ruby on Rails, por mencionar solo algunos ejemplos.



JSON.

Visualizador de JSON en navegadores basados en Chromium:

<https://chromewebstore.google.com/detail/jsonvue/chklaanhfefbnpoihckbnefhakgolnmc>

Controladores de eventos y funciones de devolución de llamada.

Controlador de eventos (Event handler).

```
xhttp.onreadystatechange = function() {
```

1. xhttp: objeto.
2. El código de la función verifica que readyState sea igual a 4 (que describe la situación *La operación está completa*)



Las funciones del controlador de eventos se denominan funciones de devolución de llamada (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*.



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.

<https://www.youtube.com/watch?v=Jh-LUQMwtRk>

Manipulando el objeto document desde la consola.

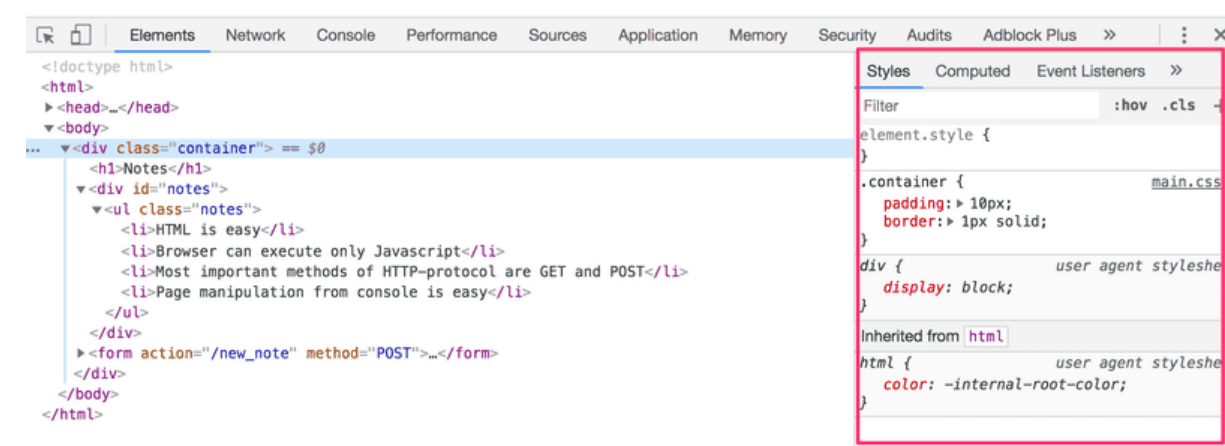


El nodo superior del árbol DOM de un documento HTML se denomina objeto *document*.



Aunque la página se actualiza en tu navegador, los cambios no son permanentes. Si se vuelve a cargar la página, la nueva nota desaparecerá porque los cambios no se enviaron al servidor.

CSS.




Ejercicios página

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.

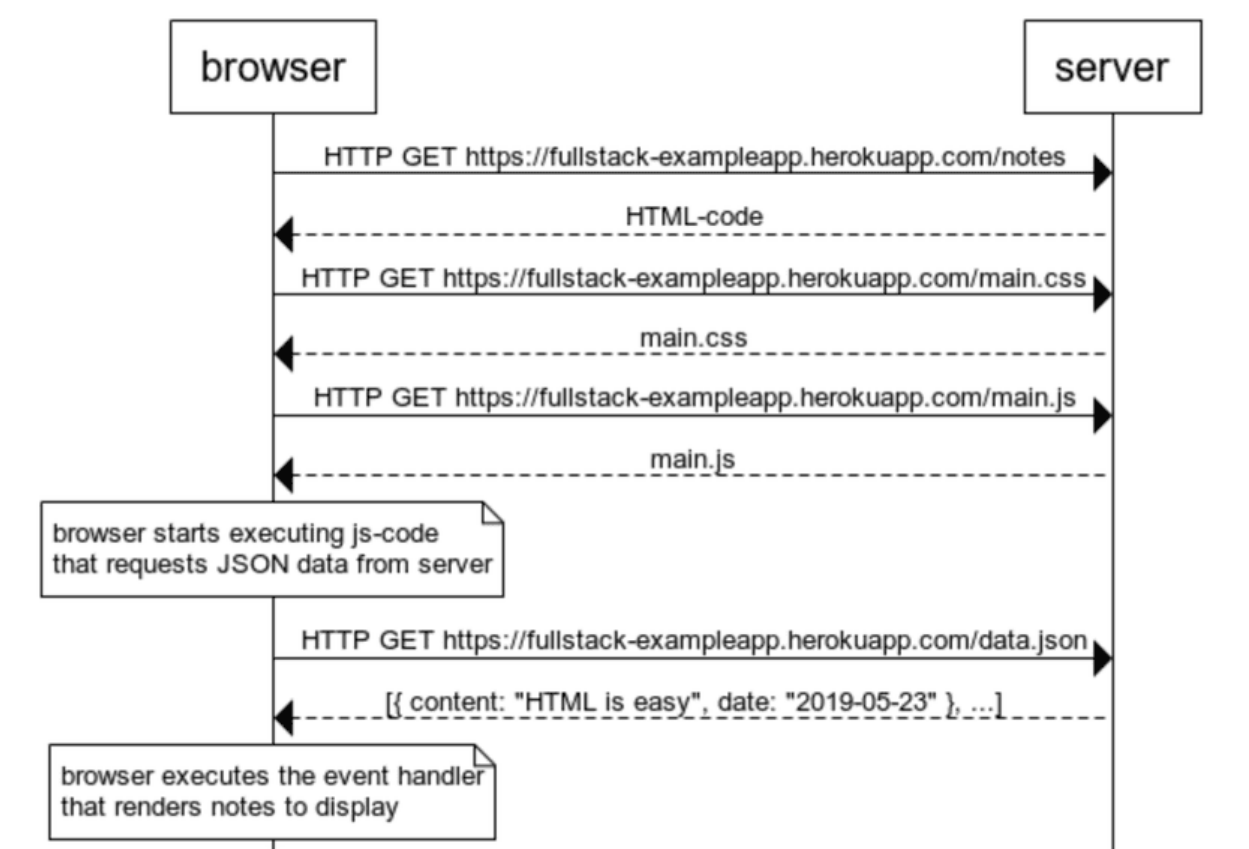
La segunda regla CSS establece el color del texto de las notas en azul.

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.



Los cambios realizados en la consola no serán permanentes. Si deseas realizar cambios duraderos, debes guardarlos en la hoja de estilos CSS del servidor.

Cargando una página que contiene JavaScript - revisión.

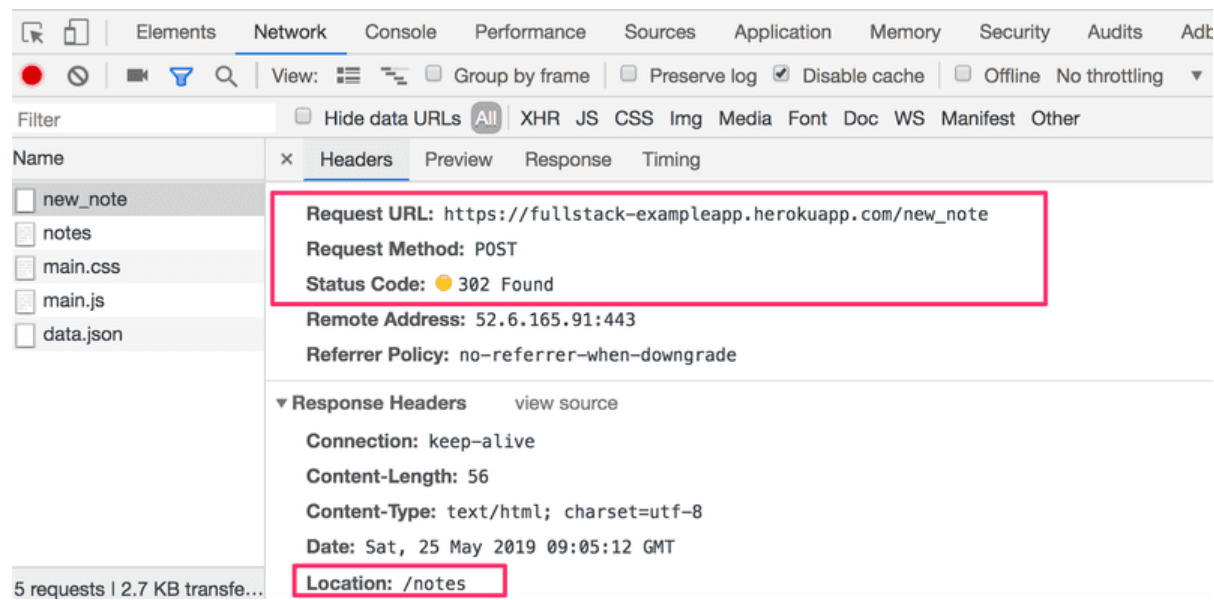


1. El navegador obtiene el código HTML que define el contenido y la estructura de la página del servidor mediante una solicitud HTTP GET.
2. Los enlaces en el código HTML hacen que el navegador también busque la hoja de estilos CSS *main.cs...*
3. ...y un archivo de código JavaScript *main.js*
4. El navegador ejecuta el código JavaScript. El código realiza una solicitud HTTP GET a la dirección <https://studies.cs.helsinki.fi/exampleapp/data.json>, que devuelve las notas como datos JSON.
5. 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.

https://developer.mozilla.org/es/docs/Learn_web_development/Extensions/Forms/Your_first_form

1. Enviar formulario: Causa no menos de *cinco* solicitudes HTTP. La primera es el evento de envío de formulario.



Solicitud: HTTP POST.

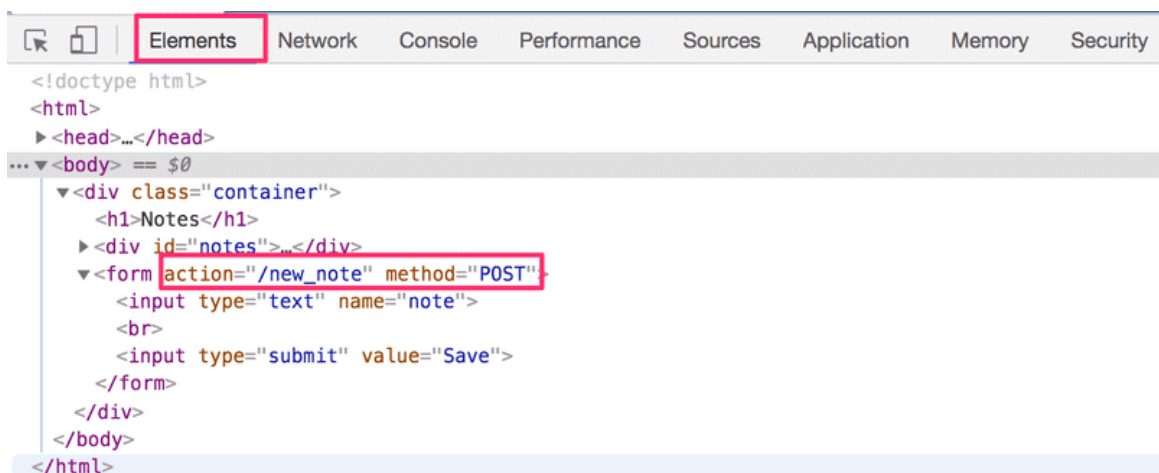
Servidor: new_note.

Estado: HTTP 302 <https://developer.mozilla.org/es/docs/Web/HTTP/Methods/POST>

Función: Redirección de URL. el servidor solicita al navegador que realice una nueva solicitud HTTP GET a la dirección definida en la *Ubicación (Location)* del encabezado - la dirección *notes*.

https://es.wikipedia.org/wiki/Redirecci%C3%B3n_de_URL

2. El navegador vuelve a cargar la página de Notas.
3. 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 *new_note*.



```
<!doctype html>
<html>
  <head>...</head>
  <body> == $0
    <div class="container">
      <h1>Notes</h1>
      <div id="notes">...</div>
      <form action="/new_note" method="POST">
        <input type="text" name="note">
        <br>
        <input type="submit" value="Save">
      </form>
    </div>
  </body>
</html>
```

4. El código en el servidor responsable de la solicitud POST (se encuentra en el servidor, no en el navegador).

```
app.post('/new_note', (req, res) => {
  notes.push({
    content: req.body.note,
    date: new Date(),
  })

  return res.redirect('/notes')
})
```

- 1. Los datos se envían como el cuerpo de la solicitud POST.
- 2. El servidor puede acceder a los datos accediendo al campo `req.body` del objeto `req` de la solicitud.
- 3. El servidor crea un nuevo objeto de nota y lo agrega a un arreglo llamado `notes`.


```
notes.push({
  content: req.body.note,
  date: new Date(),
})
```

- 1. Los objetos `note` tienen dos campos: `content` que contiene el contenido real de la nota y `date` que contiene la fecha y hora en que se creó la nota.
- 2. El servidor no guarda nuevas notas en una base de datos, por lo que las nuevas notas desaparecen cuando se reinicia el servidor.

AJAX.

<https://es.wikipedia.org/wiki/AJAX>

Es una técnica de desarrollo web para crear aplicaciones web asíncronas. Estas aplicaciones se ejecutan en el cliente, es decir, en el navegador de los usuarios mientras se mantiene la comunicación asíncrona con el servidor en segundo plano. De esta forma es posible interactuar con el servidor sin necesidad de recargar la página web, mejorando la interactividad, velocidad y usabilidad en las aplicaciones.



Hoy en día, URLs como estas no se consideran aceptables, ya que no siguen las convenciones generalmente reconocidas de las APIs RESTful.

REST.

En la actualidad se usa en el sentido más amplio para describir cualquier interfaz entre sistemas que utilice directamente HTTP para obtener datos o indicar la ejecución de operaciones sobre los datos, en cualquier formato (XML, JSON, etc) sin las abstracciones adicionales de los protocolos basados en patrones de intercambio de mensajes, como por ejemplo SOAP.

Aplicación de una sola página.

Toda la lógica está en el servidor y el navegador solo muestra el HTML como se indica.

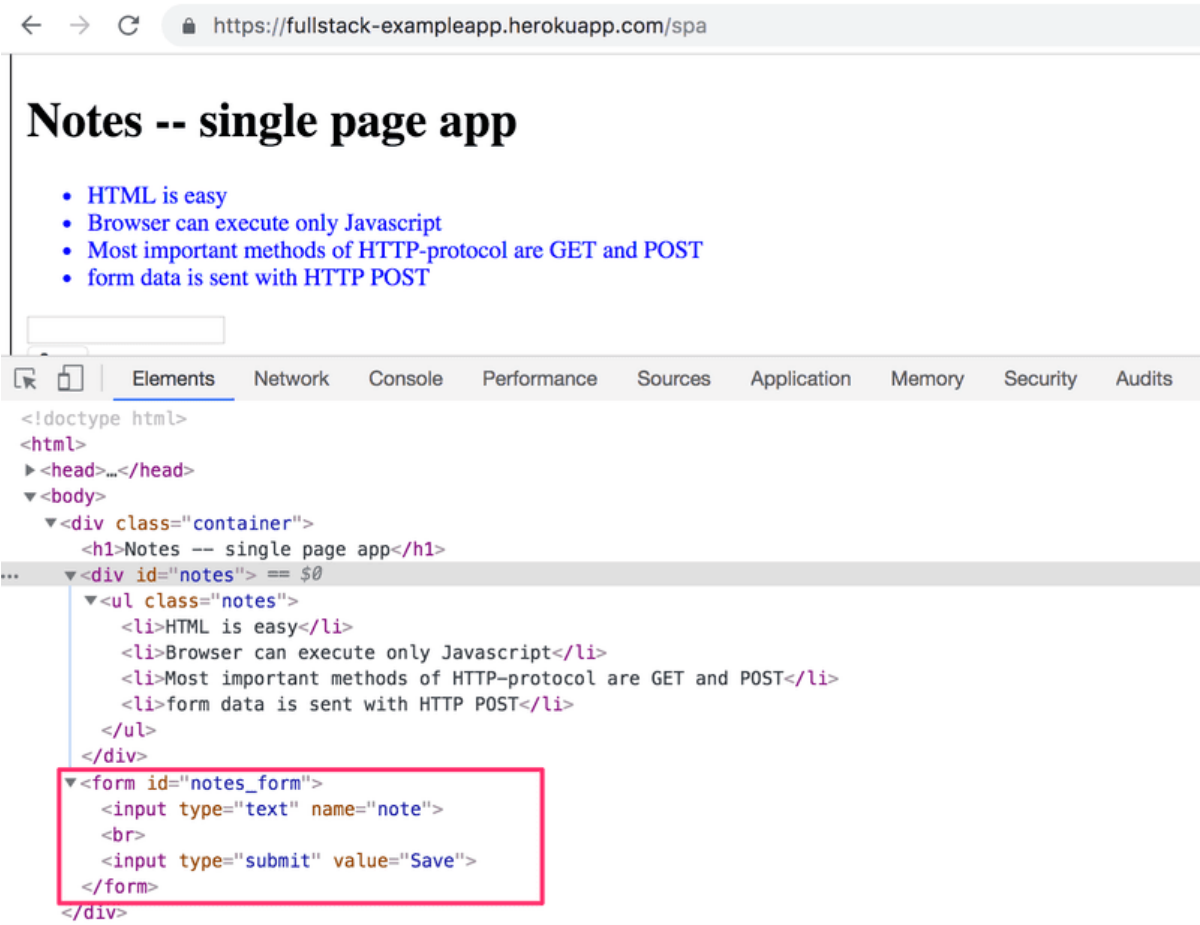
La página Notas da parte de la responsabilidad al navegador, la generación del código HTML para las notas existentes. El navegador aborda esta tarea ejecutando el código JavaScript que obtuvo del servidor. El código obtiene las notas del servidor como datos JSON y agrega elementos HTML para mostrar las notas en la página usando la DOM-API.

SPA.

En los últimos años, ha surgido el estilo de Aplicación de una sola página (SPA) para crear aplicaciones web. Los sitios web de estilo SPA no obtienen todas sus páginas por separado del servidor como lo hace nuestra aplicación de muestra, sino que comprenden solo una página HTML obtenida del servidor, cuyo contenido se manipula con JavaScript que se ejecuta en el navegador.

Aunque la lógica para representar las notas se ejecuta en el navegador, la página sigue utilizando la forma tradicional de agregar nuevas notas. Los datos se envían al servidor con el envío del formulario, y el servidor indica al navegador que vuelva a cargar la página Notas con un *redireccionamiento*.

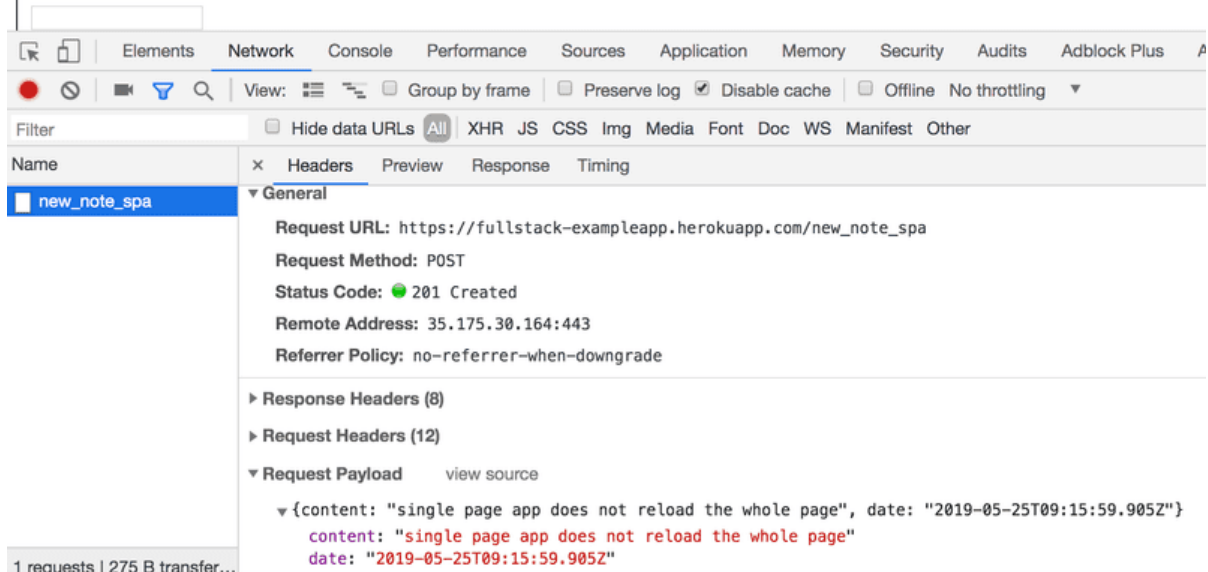
Página SPA: <https://studies.cs.helsinki.fi/exampleapp/spa>



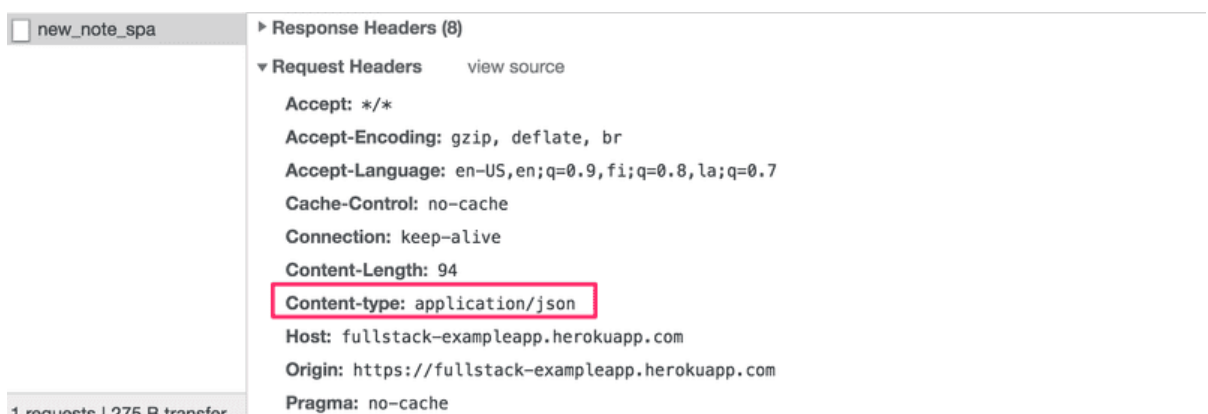
1. El formulario no tiene atributos de *action* o *method* para definir cómo y dónde enviar los datos de entrada.

Notes -- single page app

- HTML is easy
- Browser can execute only Javascript
- Most important methods of HTTP-protocol are GET and POST
- form data is sent with HTTP POST
- single page app does not reload the whole page



2. La solicitud POST a la dirección *new_note_spa* contiene la nueva nota como datos JSON que contienen tanto el contenido de la nota (*content*) como la marca de tiempo (*date*):



3. La cabecera *Content-Type* de la solicitud le dice al servidor que los datos incluidos están representados en formato JSON.
4. Sin esta cabecera, el servidor no sabría cómo analizar correctamente los datos.
5. 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.
6. 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.

Repositorio del código:

https://github.com/mluukkai/example_app

Librerías JavaScript.

Herramientas:

<https://backbonejs.org/> #La primer herramienta "moderna".

<https://angularjs.org/> #De Google.

<https://react.dev/> #El usado en el curso, de Facebook.

<https://vuejs.org/> #Está en crecimiento.

Librerías:

<https://jquery.com/> #Está algo desactualizado, pero siguen utilizándolo.

<https://github.com/reduxjs/redux>

Desarrollo Web Full Stack.

Prácticamente todas las aplicaciones web tienen (al menos) dos "capas": el navegador, al estar más cerca del usuario final es la capa superior, y el servidor la inferior. A menudo también hay una capa de base de datos debajo del servidor. Por lo tanto, podemos pensar en la *arquitectura* de una aplicación web como una especie de *stack (pila)* de capas.

A menudo, también hablamos sobre el frontend y el backend. El navegador es el frontend y el JavaScript que se ejecuta en el navegador es el código del frontend. El servidor, por otro lado, es el backend.

En el contexto de este curso, el desarrollo web full stack significa que nos enfocamos en todas las partes de la aplicación: el frontend, el backend y la base de datos. A veces, el software del servidor y su sistema operativo se ven como parte del stack, pero no vamos a entrar en ellos.

Fatiga de JavaScript.

Comunicarse en la web requiere conocimientos del protocolo HTTP. También se deben manejar las bases de datos y la administración y configuración del servidor. También sería bueno saber suficiente CSS para hacer las aplicaciones al menos algo presentables.

<https://medium.com/@ericclemmons/javascript-fatigue-48d4011b6fc4>

<https://auth0.com/blog/how-to-manage-javascript-fatigue/>

Diagrama de secuencias:

<https://www.geeksforgeeks.org/unified-modeling-language-uml-sequence-diagrams/>

Del video:

[https://www.youtube.com/watch?](https://www.youtube.com/watch?v=wTpuKOhGfJE&list=PLV8x_i1fqBw0Kn_fBIZTa3wS_VZAqddX7)

[v=wTpuKOhGfJE&list=PLV8x_i1fqBw0Kn_fBIZTa3wS_VZAqddX7](https://www.youtube.com/watch?v=wTpuKOhGfJE&list=PLV8x_i1fqBw0Kn_fBIZTa3wS_VZAqddX7)

29:37: Niveles de importancia de peticiones al servidor.

Status code:

https://es.wikipedia.org/wiki/Anexo:C%C3%B3digos_de_estado_HTTP

<https://http.cat/>

Ejercicios 0.1 a 0.3.

```
sequenceDiagram
```

```
    participant browser
```

```
    participant server
```

```
    %% Paso 1: El usuario escribe y hace clic en "Save"
```

```
    browser->>browser: User writes a note and clicks on "Save".
```

```
    %% Paso 2: El navegador envía la solicitud HTTP POST con los datos del
```

```
    browser->>server: HTTP POST https://studies.cs.helsinki.fi/exampleapp/n
```

Note right of server: Server receives form data (req.body.note)

%% Paso 3: El servidor procesa la solicitud y responde con una redirección
activate server
server-->>browser: HTTP 302: https://studies.cs.helsinki.fi/exampleapp/
deactivate server

%% Paso 4: El navegador realiza una solicitud GET para cargar la página
browser->>server: HTTP GET https://studies.cs.helsinki.fi/exampleapp/no
activate server
server-->>browser: HTML-code
deactivate server

%% Paso 5: El navegador solicita los recursos (CSS, JS y datos de las notas)
Note right of server: Browser requests CSS, JS and note data resources
browser->>server: HTTP GET https://studies.cs.helsinki.fi/exampleapp/main.css
activate server
server-->>browser: main.css
deactivate server

browser->>server: HTTP GET https://studies.cs.helsinki.fi/exampleapp/main.js
activate server
server-->>browser: main.js
deactivate server

browser->>server: HTTP GET https://studies.cs.helsinki.fi/exampleapp/data.json
activate server
server-->>browser: [{ "content": "Hello", "date": "2024-12-29" }, ...]
deactivate server

%% Paso 6: El navegador renderiza las notas (incluyendo la nueva nota)
Note right of browser: Browser updates interface and displays new note

sequenceDiagram

participant browser
participant server

%% Paso 1: El usuario accede a la versión SPA
browser->>server: HTTP GET https://studies.cs.helsinki.fi/exampleapp/spa
activate server
server-->>browser: HTML code (SPA version)
deactivate server

%% Paso 2: El navegador solicita el archivo CSS
browser->>server: HTTP GET https://studies.cs.helsinki.fi/exampleapp/main.css
activate server
server-->>browser: main.css
deactivate server

%% Paso 3: El navegador solicita el archivo JavaScript spa.js
browser->>server: HTTP GET https://studies.cs.helsinki.fi/exampleapp/spa.js
activate server
server-->>browser: spa.js
deactivate server

%% Paso 4: El navegador solicita los datos JSON de las notas

```
browser->>server: HTTP GET https://studies.cs.helsinki.fi/exampleapp/da
activate server
server-->>browser: [{ "content": "Note 1", "date": "2024-12-29" }, ...
deactivate server
```

%% Paso 5: El navegador renderiza las notas
Note right of browser: Browser processes JavaScript and updates the UI

sequenceDiagram

participant browser
participant server

%% Paso 1: El usuario escribe una nota y hace clic en "Save"
browser->>browser: User writes a note and clicks "Save".
Note right of browser: Browser executes JavaScript event handler.

%% Paso 2: El navegador prepara la nueva nota en formato JSON
browser->>browser: Create JSON { "content": "New note", "date": "2024-12-29" }

%% Paso 3: El navegador envía la solicitud POST con la nueva nota
browser->>server: HTTP POST https://studies.cs.helsinki.fi/exampleapp/new_note
activate server
Note right of server: Server receives JSON data.

%% Paso 4: El servidor responde con 201 Created
server-->>browser: HTTP 201 Created
deactivate server

%% Paso 5: El navegador actualiza la interfaz sin recargar
browser->>browser: Add new note to the list and re-render UI.
Note right of browser: Notes list updated dynamically using DOM manipulation