FrontEndII
Profesor: Martín Bonino

Módulo 5. Asincronismo y API

Sesión 16. Introducción a Asincronismo
Abril 5 de 2022

Ajax
AJAX (Asynchronous JavaScript and XML) es un conjunto de tecnologías que se utilizan para crear aplicaciones web asíncronas. Esto las vuelve más rápidas y con mejor respuesta a las acciones del usuario.

¿Cómo funciona?
1. Se produce un evento en una página web (se carga la página, se hace clic en un botón).
2. JavaScript crea un objeto XMLHttpRequest.
3. El objeto XMLHttpRequest envía una solicitud a un servidor web.
4. El servidor procesa la solicitud.
5. El servidor envía una respuesta a la página web.
6. La respuesta es leída por JavaScript.
7. JavaScript realiza la acción adecuada (como la actualización de la página).

Ejemplos:
- Autocompletado de Goolge.
- Chat de atención al cliente.
- Notificación en redes sociales.

HTTP: Petición – Respuesta

HTTP o Hypertext Transfer Protocol es un protocolo de intercambio de datos en la Web entre cliente y servidor. Los mensajes HTTP forman una estructura como medio para realizar una petición de datos iniciada por el cliente, normalmente un navegador web, en busca de su respuesta ejecutada por el servidor.

Asincronismo
JavaScript es un lenguaje de programación asíncrono porque es capaz de ejecutar un hilo de tareas o peticiones en las cuales, si la respuesta demora, el hilo de ejecución de JavaScript continuará con las demás tareas que hay en el código. Gracias al asincronismo, se genera un código más funcional, rápido y eficiente, sin necesidad de recargar la página para actualizar la información.

Concurrencia
cuando las tareas pueden comenzar, ejecutarse y completarse en períodos de tiempo superpuestos, en donde al menos dos hilos están progresando.

Paralelismo
Cuando dos o más tareas se ejecutan exactamente al mismo tiempo.

Campo Request
tienen 3 componentes:
- Línea de inicio: Método HTTP + Path + Versión del protocolo
- Cabeceras: Son opcionales
- Cuerpo

Campo Response
También tienen tres componentes:
- Línea de estado: Versión del protocolo + Código de estado + descripción
- Cabeceras
- Cuerpo

Introducción a HTTP:
HyperText Transfer Protocol, o en castellano: protocolo de transferencia de hipertexto, es un protocolo que sirve para gestionar la comunicación entre dos máquinas conectadas a una red, donde una de las máquinas solicita un contenido específico (request) y la otra se encarga de responder a dicha solicitud (response).

HTTP permite navegar hacia sitios web a través de direcciones www y enlaces.

¿Qué es una URI?
El protocolo HTTP permite la transferencia de información en la web a través de direcciones web, técnicamente llamadas URI. Una URI (identificador de recursos uniformes) es un bloque de texto que se escribe en la barra de direcciones de un navegador web y está compuesto por dos partes: la URL y la URN.

La URL indica dónde se encuentra el recurso que se desea obtener y comienza siempre con un protocolo.
La URN es el nombre exacto del recurso uniforme, incluyendo el nombre del dominio.

Métodos de Petición:
- Get. Para solicitar datos. Cuando se escribe una dirección en el navegador o se accede a un enlace, se está utilizando el método GET. En caso de querer enviar información al servidor usando este método, la misma viajará a través de la URL.
- Post. Para enviar datos al servidor. Este método es más seguro que GET, ya que la información no viaja a través de la URL.
- Put. Se usa para reemplazar información actual
- Patch: Para aplicar modificaciones parciales a un recurso
- Delete: Borrar un recurso presente en el servidor

Códigos de estado HTTP:
Cada vez que el servidor recibe una petición o request, este emite un código de estado que indica, de forma abreviada, el estado de la respuesta HTTP. El código tiene tres dígitos. El primero representa uno de los 5 tipos de respuesta posibles:

1: Respuestas informativas
2: Respuestas exitosas
3: Redirecciones
4: Errores del cliente
5: Errores de servidor

Ejemplos:

200: OK → La petición se realizó con éxito.
301: Moved Permanently → El recurso se ha movido.
302: Found → El recurso fue encontrado.
304: Not Modified → El recurso no cambió, se cargará desde el caché.
400: Bad Request → El pedido está mal.
401: Unauthorized → No estás autorizado, seguramente debas autenticarte.
403: Forbidden → El pedido está prohibido y no debería repetirse.
404: Not Found → El recurso no fue encontrado.
500: Internal Server Error → Hubo un error en el servidor.
503: Service Unavailable → El servicio solicitado no está disponible.
550: Permission denied → Permiso denegado.

HTTPS es una versión mejorada del protocolo HTTP. Usando este protocolo, el servidor codifica la sesión con un certificado digital.

Sincrónico

Promise es un objeto nativo de JavaScript desde ECMA6.

Let miPrimeraPromise = new Promise((resolve, reject) => {

async, await. Consultar

Sesión 17. API 1
Abril 7 de 2022

API: Application Programming Interface
Es una interfaz que permite la comunicación entre 2 aplicaciones.

RESTCountries: Devuelve información sobre todos los países del mundo.

Endpoint:
Es un punto de conexión al que se debe apuntar para obtener la información que se desea. Son las URL que se deben utilizar para obtener información de un servidor a través de una API.

REST: Representational State Transfer
Es un tipo de arquitectura de servicios que proporciona estándares entre sistemas informáticos para establecer cómo se van a comunicar entre sí.
Un sistema REST busca implementar un esquema o protocolo que le permita a todos los sistemas que se comunican con él entender en qué forma lo tienen que hacer y bajo qué estructura deberán enviar sus peticiones para que sean atendidas.

Es una arquitectura del tipo cliente-servidor porque debe permitir que tanto la aplicación del cliente como la aplicación del servidor se desarrollen o escalen sin interferir una con la otra.

Una arquitectura REST expone a los clientes a una interfaz uniforme:

- Todos los recursos del servidor tienen un nombre en forma de URL o hipervínculo.
- Toda la información se intercambia a través del protocolo HTTP.

A esas URL se les llama endpoints, es decir, el servidor expone a los clientes un conjunto de endpoints para que este pueda acceder. A esa interfaz uniforme, o sea, al conjunto de endpoints, se le llama API.

Características:

1. Permite separar la aplicación web en dos partes: la interfaz de usuario y todo aquello que la aplicación provee como servicio que la interfaz cosume (?). Existirían unas url que se encargan de la interfaz y otras url que se encargan de proveer la información. Las capas se conectan mediante el uso de API.

2. La ubicación de los recursos. Existe una ruta única que define la ubicación de cada recurso específico.

3. Stateless (Sin estado). El servidor no almacena ningún dato acerca de las peticiones que hace el cliente. De esta forma, cada solicitud se trata de forma independiente. No existe el concepto de sesión de usuario. Por ejemplo, si se

desea que el servidor distinga entre usuarios logueados o invitados, se debe mandar toda la información de autenticación necesaria en cada petición que se le haga a dicho servidor. Esto permite desarrollar aplicaciones más confiables, performantes y escalables.

4. Cacheable. El cacheo de datos se implementa del lado del cliente, para mejorar la performance y reducir la demanda al servidor.

Formatos admisibles:
- JSON. En la cabecera debe especificar el formato. Ej: 'Content-Type: 'application/json'.
- Raw: La información viaja como texto plano, y por lo tanto es poco usado.
- Text: Permite enviar datos que no tienen formato JSON, por ejemplo código html.
- URL-encoded: La infromación viaja como texto codificado, similar a un query string.

Método HEAD: Permite verificar la última fecha de modificación de un recurso.

AJAX Fetch
Fetch es una función nativa que permite hacer pedidos a una API desde el front-end.

Promesa
then

fetch()
Recibe como primer parámetro la URL del endpoint al cual se hace el llamado asincrónico. Al no saber cuándo se completa la petición, el servidor devuelve una promesa.

```
fetch("https://restcountries.eu/rest/v2/")
.then(function(response){
	return response.json();
})
.then(function(data){
	console.log(data)
})
.catch(function(error){
	console.error(error)
})
```

Manejo de errores

Los errores que se producen en un programa pueden ocurrir debido a un descuido del programador, una entrada inesperada del usuario, una respuesta errónea del servidor, entre otras razones. Por lo general, un script es interrumpido y se detiene cuando esto sucede. Pero podemos evitarlo con try...catch que permite "atrapar" errores para que el script pueda funcionar igualmente.

- La declaración **try** permite probar un bloque de código en busca de errores.
- La declaración **catch** permite manejar el error.
- La declaración **throw** permite crear errores personalizados.
- La declaración **finally** permite ejecutar código, después de intentar y capturar, independientemente del resultado.

Sintaxis:

```
try {
	// Block of code to try
}
catch(err) {
	// Block of code to handle errors
}
finally {
	// Block of code to be executed regardless of the try / catch result
}
```

Ejemplo

```
function myFunction() {

	let message, x;
	message = document.getElementById("intro");
	message.innerHTML = "";

	x = document.getElementById("demo").value;

	//Se ejecuta un try con condicionales arrojando un mensaje:
	try {
		if(x == "") throw "Contenido vacio";
		if(isNaN(x)) throw "No es un numero";
		x = Number(x);
		if(x > 10) throw "Numero demasiado alto";
		if(x < 5) throw "Numero demasiado bajo";
	}

	//Se ejecuta un catch para manejar el error mostrándolo en
```

```
        el navegador
        catch(err) {
                message.innerHTML = "Error: " + err + ".";
        }

        //Se ejecuta la accion que termina con la funcion para devolver el valor
        requerido
        finally {
                document.getElementById("demo").value = "";
        }
}
```

Tipos de errores:

- RangeError: Número fuera de rango
- ReferenceError: Referencia ilegal
- Error de sintaxis:
- Error de tecleado
- URIError: error en encodeURI ().

Preguntas

- ¿Por qué se requieren 2 then?
- Diferencia entre los tipos de formatos. ¿Cuándo se usa form-data?
- Uso del método HEAD
- Headers. ¿Qué especifican?

Sincrónico

- dog API: Devuelve imágenes de perros
- API con la cotización del dólar
- API con contenido falso de usuarios
- API de Twitter.

Sesión 18. API 2
Abril 8 de 2022

AJAX Fetch – POST

Sintaxis:

```
fetch(url, {
        method: “POST”,
        body: JSON.stringify(data),
        headers: {
                “Content-Type”: “application/json”
        }
})
.then(function(res) {
        return res.json();
})
.then(function(data) {
        console.log(data);
})
.catch(function(err) {
        console.log(“Error! “ + err);
})
```

Ejemplo:
En ocasiones la API suministra una clave de acceso que se debe incluir en los encabezados:

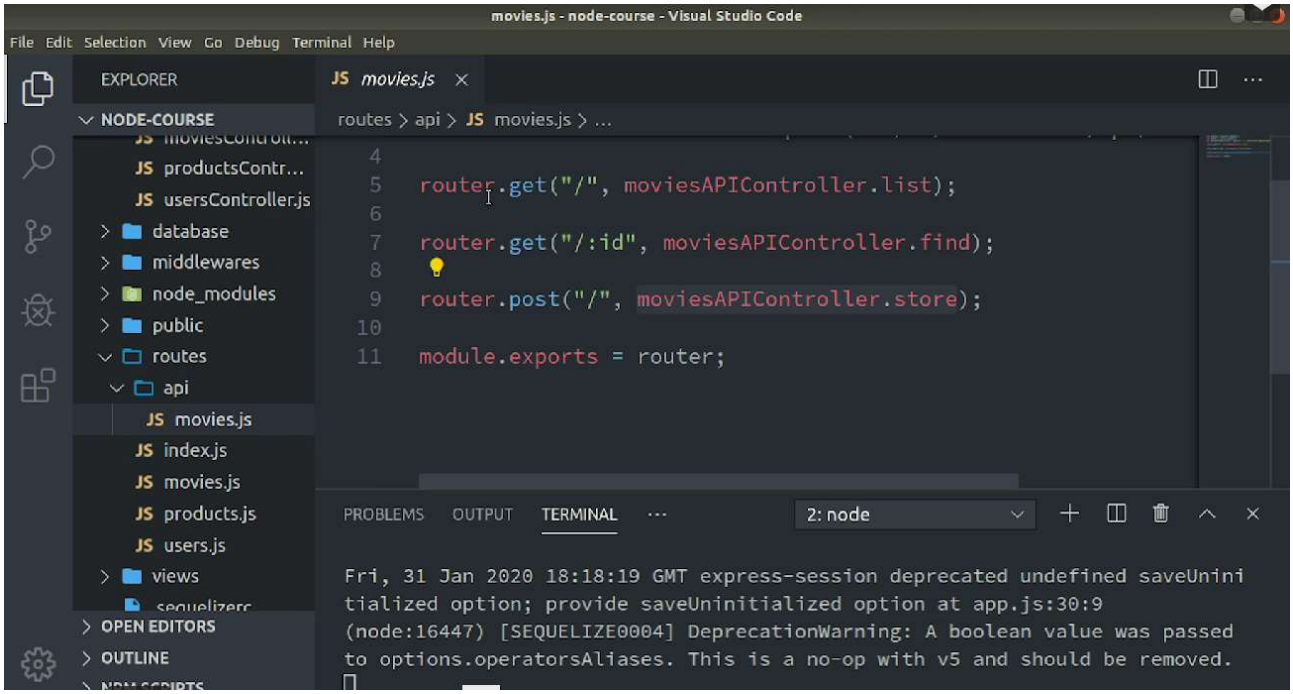
```
headers: {
        “Content-Type”: “application/json”,
        “x-api-key”: “89d0ffa5-5d01-42d0-b830-ab27347edad”
}
```

Postman

Clientes HTTP: Navegadores, Postman.

Ejemplo
En el ejemplo, se envían los datos de la petición POST en el body, con formato x-www-form-urlencoded. (Consultar)

Ejemplo
Ver API de normalización de datos geográficos de Argentina:
https://apis.datos.gob.ar/georef/api/provincias



Preguntas

- Diferencia entre los tipos de formatos. ¿Cuándo se usa form-data?
- Uso del método HEAD
- Headers. ¿Qué especifican?
- Validación de formaularios en HTML

Sincrónico

API: JSONPlaceholder.

Datasets en html

Formato raw: crudo.

En Postman: Enviar con raw, formato JSON.

Sesión 19. To-Do App: docs
Abril 12 de 2022

La API puede devolver dos tipos de respuesta:

Respuesta exitosa:
Cuando la petición ha sido procesada satisfactoriamente. Devuelve un status 2XX.

Error:
Cuando la petición no puede procesarse debido a algún error. Devuelve un status 4XX si se trata de un error en la petición, o un status 5XX si el error es del servidor.