CURSO .JS

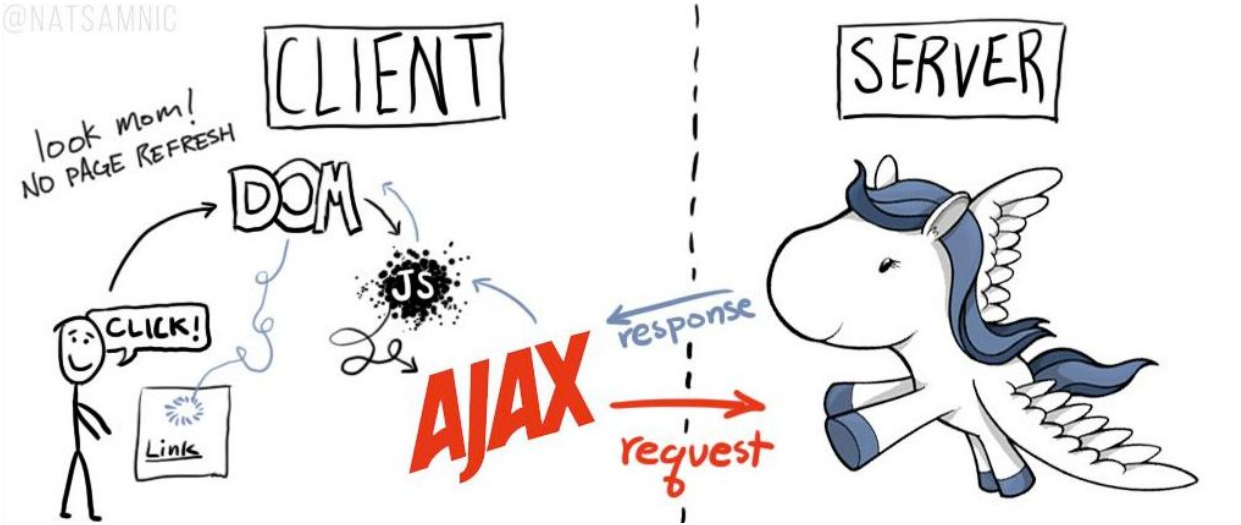
Comunicación con el servidor

Autor: Jon Vadillo
www.jonvadillo.com

AJAX

"Técnica de desarrollo web para crear aplicaciones interactivas, donde el cliente (navegador) mantiene la comunicación asíncrona con el servidor en segundo plano"

AJAX


@NATSAMNIC
CLIENT
SERVER
look mom!
NO PAGE REFRESH
DOM
JS
CLICK!
Link
AJAX
response
request

AJAX

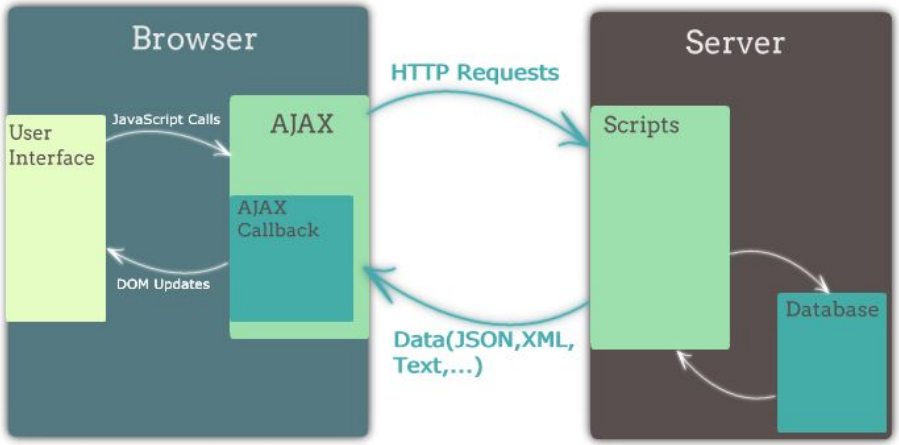


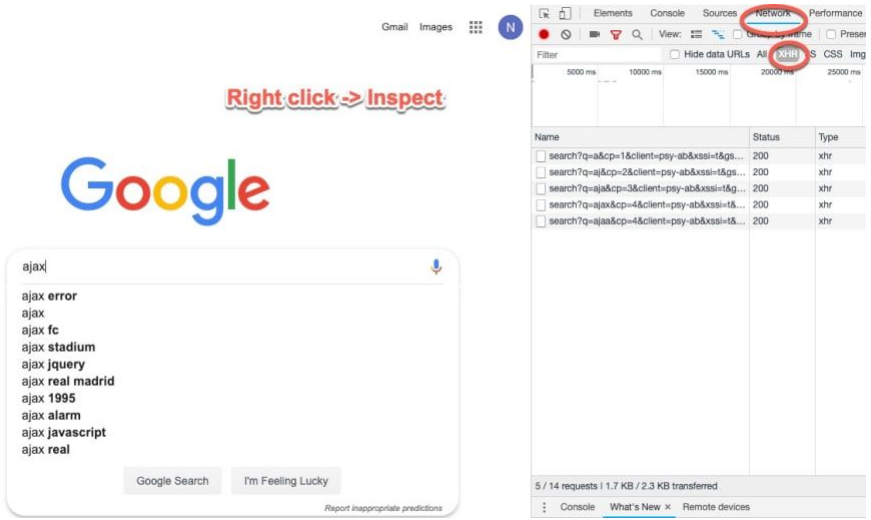
Image source: http://javascript-coder.com

AJAX

- Es una técnica que permite crear páginas web que actualicen su contenido **sin necesidad de recargar la página** completa.
- La página se **comunica con el servidor de forma asíncrona** (mediante JS) y cuando recibe la respuesta del servidor la procesa. Es decir, una vez lanza la petición al servidor, puede seguir realizando otras tareas hasta que recibe la respuesta.
- Desde que se introdujo en 2005 ha evolucionado
    - El intercambio de datos de forma asíncrona se realiza mayormente mediante **JSON** y no XML.

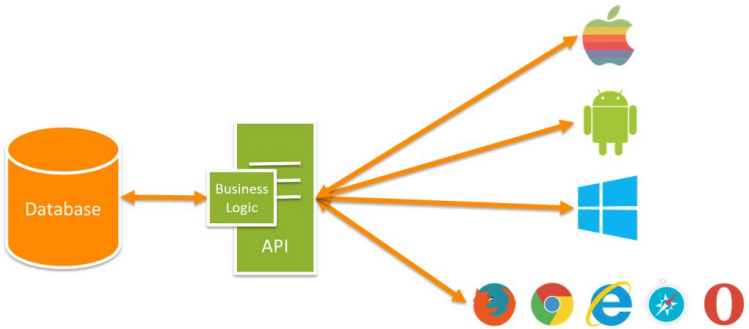
Ejemplo de usos

- Carga de contenido con scroll vertical infinito.
- Validación de email existente en la base de datos mientras se completa un formulario.
- Cajas de texto con sugerencias automáticas
- etc.



API (Application Programming Interface)

- Conjunto de **reglas que describen cómo una aplicación puede interactuar con otra**.
- Define los métodos que un software puede utilizar para interactuar con otro.
- Pueden ser Web APIs u otro tipo de APIs, como las del kernel de Linux o una comunicación por Bluetooth.
    - **Todos los servicios Web son API, pero todas las API no son servicios Web**.



Servicio Web (Web Service)

- API que tiene **HTTP** como protocolo de comunicación.
- Predominan dos tipos de servicios web:
  - **SOAP** (Simple Object Access Protocol): exponer operaciones individuales (servicios)
  - **REST** (Representational State Transfer): se centra en **operaciones basada en recursos**, heredando las operaciones HTTP (GET, PUT, UPDATE, DELETE, POST).
- El intercambio de información se puede hacer en distintos formatos: **XML** y **JSON**

JSON (JavaScript Object Notation)

- **Formato** para el intercambio de datos.
- Aunque es muy parecido a la sintaxis de objeto literal de JavaScript, puede ser utilizado independientemente de JavaScript.
- Importante
  - Requiere **dobles comillas en la clave**.
  - **Contiene solo propiedades**, no métodos
- Validador online: https://jsonlint.com/

```
{
    "nombre": "Markel Jainaga",
    "edad": 29,
    "email": "markel@email.com",
    "aficiones": [
      "Lectura",
      "Baloncesto",
      "Informática"
    ]
 }
```

REST Web Service

- **Protocolo cliente/servidor sin estado**: cada petición HTTP contiene toda la información necesaria para ejecutarla, lo que permite que ni cliente ni servidor necesiten recordar ningún estado previo (sesiones).
- **Aplica acciones concretas** sobre los recursos identificados con una URI.
- **Las operaciones más importantes** son cuatro: POST (crear), GET (leer y consultar), PUT (editar) y DELETE (eliminar).
- La API REST es independiente del tipo de plataformas o lenguajes

REST Web Service

URL	HTTP Verb	POST Body	Result
/api/movies	GET	empty	Returns all movies
/api/movies	POST	JSON String	New movie Created
/api/movies/:id	GET	empty	Returns single movie
/api/movies/:id	PUT	JSON string	Updates an existing movie
/api/movies/:id	DELETE	empty	Deletes existing movie

REST Web Service

Resource	GET (Read)	POST (Create)	PUT (Update)	DELETE (Delete)
/users	Returns a list of users	Creates a new user	Bulk update of users	Delete all users
/users/123	Returns a specific User	Method not allowed (405)	Updates a specific user	Deletes a specific user

REST URL naming

- Utiliza nombres: REST se refiere a recursos (nombres), no acciones (verbos):
  - Bien: http://api.example.com/users
  - Mal: http://api.example.com/mostrarUsuarios
- Sé consistente con la jerarquía:
  - http://api.example.com/users/{id}/posts
  - http://api.example.com/posts/{id}/comments
- Utiliza parámetros (y no nuevas URIs) para filtrar, ordenar,..
  - http://api.example.com/users?region=USA&role=admin

Respuestas HTTP

- Informational **1XX** → Difícil de encontrar
- Successful **2XX** → 200 OK (la petición se ha respondido correctamente)
- Redirection **3XX**: 308 Redirección (CSS cacheado no)
- Client Error **4XX**: 404 Not Found (la página no existe), 403 Forbiden
- Server Error **5XX**: 500 Internal Server Error (error genérico)

XMLHttpRequest

```
let xhr = new XMLHttpRequest();
xhr.open('GET', 'http://domain/service');
xhr.onreadystatechange = function() {
  // request completed?
  if (xhr.readyState !== 4) return;
  if (xhr.status === 200) {
    // request successful
    console.log(xhr.responseText);
  }
  else {
    // request error
    console.log('HTTP error', xhr.status, xhr.statusText);
  }
};
```

Fetch API

```
fetch('url-de-API')
  .then(res => {
    // tratar la respuesta
  })
  .catch(error => {
     // tratar el error
  })
```

Fetch API

- La API Fetch proporciona una interfaz para obtener recursos
  - Provee del método global **fetch()** que proporciona una forma fácil y lógica de **obtener recursos de forma asíncrona por la red**.
- Está basada en la **Promises** (promesas), técnica utilizada para la programación asíncrona.
- El método fetch() toma como **argumento obligatorio una URL**.
  - Puede aceptar un **segundo parámetro opcional**, un objeto init que **permite controlar algunos ajustes**.

Promises

- Las promesas (en inglés, *promise*) son objetos que representan la finalización (exitosa o con error) de una **tarea asíncrona** (por ejemplo un envío de datos al servidor, consulta de datos al local storage, etc.).
- A la hora de ejecutarlas, **no sabemos cuándo tendremos disponible la respuesta**, por lo que hay que posponer las operaciones que dependan de esa respuesta.
- Nos “prometen” que cuando finalice la operación que tiene asignada, nos avisará para que ejecutemos el código que queramos.
- Tienen 3 estados: **pendiente, resuelta y rechazada** (cuando se obtiene un error).

Promises

- Mientras la promesa realiza la operación asignada, **nuestro código continúa su ejecución**, pero ya hemos dejado preparado el código que queremos que se ejecute cuando la promesa finalice.
- **Cuando una promesa se resuelve**, entonces **se ejecuta el código que habíamos dejado preparado** (en forma de función que pasamos al método **.then**), ejecutando así el código que dependiente de la promesa.
- Si la promesa es rechazada entonces se ejecuta la función que pasamos a **.catch**, de esta forma podemos controlar el flujo de datos.
- **Promise chaining**: una promesa puede devolver otra promesa y llamar al siguiente .then de la cadena.

Fetch API

```
fetch('https://jsonplaceholder.typicode.com/posts/1')
  .then(response => response.json())
  .then(json => console.log(json))
```

Fetch API

```
fetch('https://reqres.in/api/users/', {
  method : 'POST',
  headers : {
    'Content-Type' : 'application/json'
  },
  body : JSON.stringify({
    name : 'Ane Oiarzabal'
  })
})
.then(response => response.json())
.then(json => console.log(json))
```

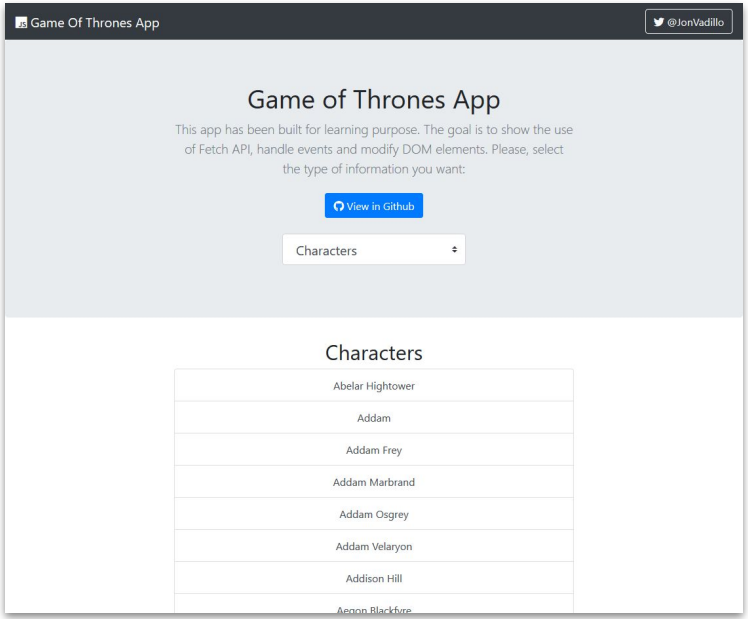
Errores 404, 500, ...

- La promesa **no se considera rechazada con un estado de error HTTP** incluso si la respuesta es un error HTTP 404 o 500. Solo será rechazada ante un fallo de red o si algo impidió completar la solicitud.

```
fetch('url')
  .then(response => {
    if (response.ok) {
      return response.json()
    } else {
      console.log(response.statusCode)
      return Promise.reject('Error en la
            respuesta del server')
    }
  })
  .then(data => console.log(data))
  .catch(error => console.log('Error: ', error));
```

Game of Thrones App

- Entra en la página https://gotapi.netlify.app/ para ver la aplicación de ejemplo.
- Accede al repositorio GitHub para ver el código JavaScript utilizado:
  - Fetch API
  - Manejo de Eventos
  - Generación de elementos HTML



Hands on!

- En esta ocasión tendrás que crear una página que utilice el API “Game of Thrones Quotes API”.
  - Entra en el repositorio https://github.com/wsizoo/game-of-thrones-quotes para ver la documentación de uso del API.
  - Crea una página que obtenga mediante la función fetch() los resultados y lo muestre por pantalla generando elementos HTML.

Sources

- Mozilla MDN: https://developer.mozilla.org/es/
- Modern JavaScript: https://javascript.info/