

# TEMA 3. (Parte 2)

## SERVICIOS WEB REST (CLIENTE)

---

Pedro Delgado Pérez, Guadalupe Ortiz Bellot,  
David Corral Plaza

Grado en Ingeniería Informática  
Departamento de Ingeniería Informática

# ESQUEMA INICIAL

**Cliente**



**Servidor**



**Base de datos**



1



2



4



3



*Index.html*

*index.js/movies.js*

*movies-service.js*



# ÍNDICE

- **Breve introducción: AJAX y jQuery**
- Invocación mediante AJAX y JQuery para HelloWorld
- Invocación servicios web REST películas
- Conexión de servicios con página web

# BREVE INTRODUCCIÓN A AJAX (i)

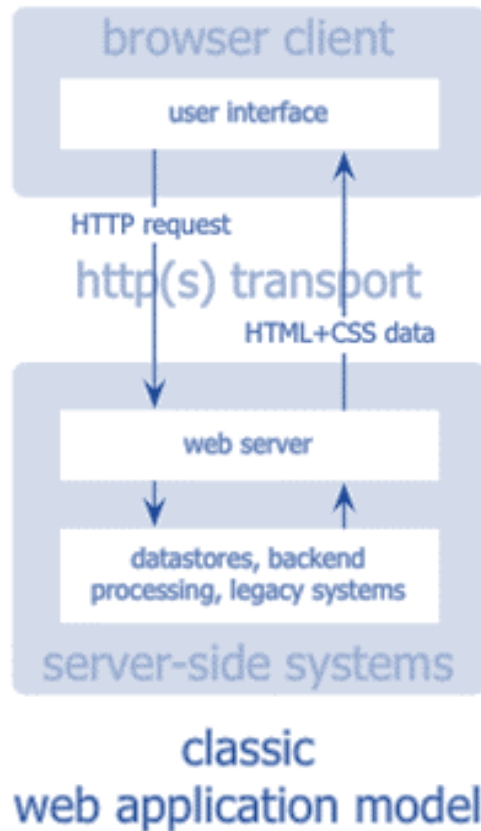
- **AJAX** (*Asynchronous JavaScript And XML*)



- Para el desarrollo de RIAs, haciéndolas mucho más rápidas, interactivas y usables.
  - Permite solicitar **transacciones HTTP desde JavaScript** (sin recarga de la página web al completo).
  - Solo se han de servir los datos necesarios para actualizar la vista.
- 
- Las tecnologías que forman una **aplicación AJAX** son:
    - *XHTML* y *CSS*, para una estructura y presentación basada en estándares.
    - *DOM*, para la interacción y manipulación dinámica de la presentación.
    - *XML, JSON...*, para el intercambio y la manipulación de información.
    - *XMLHttpRequest*, para el intercambio asíncrono de información.
    - *JavaScript*, para unir todas las demás tecnologías.

# BREVE INTRODUCCIÓN A AJAX (ii)

*Peticiones y respuestas directas entre el cliente y el servidor*



*Ajax como intermediario para lograr una transacción asíncrona*

# BREVE INTRODUCCIÓN A JQUERY (i)

## jQuery

Biblioteca cuyo objetivo es facilitar el desarrollo con JavaScript.

**Documentación:** <https://api.jquery.com/>

**Funciones:** recorrido y manipulación del DOM, control de eventos, cambio de estilos CSS, creación de animaciones,...

**Tutorial:** <https://www.w3schools.com/jquery/>

jQuery simplifica el uso de JavaScript, **en especial de AJAX:**

- Antes de ES6, soporte complejo a través de *XMLHttpRequest*.
- A partir de ES6, soporte mediante la *API Fetch*:

[https://developer.mozilla.org/es/docs/Web/API/Fetch\\_API](https://developer.mozilla.org/es/docs/Web/API/Fetch_API)

*Nosotros nos vamos a centrar en el uso de jQuery para invocar Servicios Web REST mediante AJAX*

# BREVE INTRODUCCIÓN A JQUERY (ii)

- Para usar funciones AJAX con jQuery:



`$.ajax (opciones)`

- Entre las **opciones**, indicaremos *url, data, type...*

```
1 | $.ajax({  
2 |   url: "/api/getWeather",  
3 |   data: {  
4 |     zipcode: 97201  
5 |   },  
6 |   success: function( result ) {  
7 |     $( "#weather-temp" ).html( "<strong>" + result + "</strong> degrees" );  
8 |   }  
9 | });
```

- También hay funciones simplificadas, como `$.get()` y `$.post()`, para realizar operaciones más concretas.

[https://www.w3schools.com/jquery/jquery\\_ref\\_ajax.asp](https://www.w3schools.com/jquery/jquery_ref_ajax.asp)

# ÍNDICE

- Breve introducción: AJAX y jQuery
- **Invocación mediante AJAX y JQuery para HelloWorld**
- Invocación servicios web REST películas
- Conexión de servicios con página web



# PASO 1 – CREAR ESTRUCTURA

1. Crear una **carpeta** para nuestra parte cliente (por ejemplo: *HelloWorldCliente*)
2. Crear un **fichero** “*index.html*” dentro de esa carpeta.
3. Crear la **carpeta** “*js*” dentro de esa carpeta.
4. Crear un **fichero** llamado “*wsinvocations.js*” dentro de la carpeta “*js*”.

```
pedro@pedro-espada:~/Documentos/PNET/Curso2022-23/HelloWorldCliente$ ls
index.html  js  Leeme.txt
pedro@pedro-espada:~/Documentos/PNET/Curso2022-23/HelloWorldCliente$ ls js
jquery-3.6.4.min.js.js  wsinvocations.js
```

**Nota:** en el siguiente paso, descargaremos la biblioteca jquery.

# PASO 2 – EDITAR INDEX.HTML

- Abrimos el **fichero** “*index.html*” y pegamos el siguiente código.

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Hello world!</title>
    <script type="text/javascript" src="js/jquery-3.6.4.min.js">
</script>
    <script type="text/javascript" src="js/wsinvocations.js"> </script>
</head>
<body>
    <h1>Hello World Client</h1>
</body>
</html>
```

## Incluimos dos ficheros JavaScript:

- La biblioteca *jQuery*.
- El script desde donde se realizarán las invocaciones Ajax al servidor.

# PASO 3 – DESCARGAR JQUERY

1. Accedemos al siguiente enlace (versión minimizada):

<https://code.jquery.com/jquery-3.6.4.min.js>

2. Guardamos el contenido (Guardar cómo) en un **fichero** llamado “*jquery-3.6.4.min.js*” dentro de la **carpeta** “js”.

**Nota:** Una alternativa es enlazar la URL directamente en el código:

```
<script type="text/javascript"  
  src=https://code.jquery.com/jquery-3.6.4.min.js  
  integrity="sha256-oP6HI9z1XaZNBrJURtCoUT5SUnxFr8s3BzRl+cbzUq8="<br>  
  crossorigin="anonymous">  
</script>
```

Sacar de : <https://code.jquery.com/jquery/>

**¿Y qué ventaja tiene?** La caché identifica los recursos por URL (si fue visitada previamente, el contenido se sirve de forma directa).

# PASO 4 - IMPLEMENTACIÓN

- En “*index.html*” creamos un **botón** que invocará a la función *getHello*, y un encabezado vacío con identificador *resGetHello*:

**ESTO NO ESTÁ AÑADIDO EN EL CÓDIGO PROPORCIONADO**

```
<input type="button" value="GET HELLO" onclick="getHello()" />
<h3 id="resGetHello"></h3>
```

- Crear la **función** *getHello* en “*wsinvocation.js*”:

```
function getHello() {
  $.ajax({
    type: "GET",
    url: "http://localhost:8080/",
    success: function(data) {
      $("#resGetHello").html(data); },
    error: function(res) {
      alert("ERROR: " + res.statusText); }
  });
}
```

Petición tipo GET a localhost:8080

Función *callback* que se ejecuta cuando la solicitud tiene éxito (código estado 2xx)

Función *callback* para el caso de error

# PASO 5 – RESULTADO 1

1. Ejecutamos nuestro proyecto *HelloWorld* de NodeJs que realizamos en la anterior sesión:

```
node index.js
```

2. Abrimos el **fichero** “*index.html*” de *HelloWorldCliente* con un navegador.

3. Pulsamos sobre el botón.

**¿DETECTAS FALLOS?**  
**ASEGÚRATE QUE EL CORS ESTÉ**  
**HABILITADO EN EL SERVIDOR NODEJS.**



Hemos visto el caso de una petición de origen cruzado. A continuación, en el paso 6 será el servidor quien sirva el contenido mediante *path*:


```
const path = require('path');
```

# PASO 6 – REUNIÓN DE FICHEROS

1. Crear una **carpeta** de nombre “*public*” dentro del proyecto *HelloWorld*. Mover allí todo el contenido de *HelloWorldCliente*.
2. Añadimos la siguiente línea a nuestro fichero “*index.js*”:  

```
app.use(express.static(path.join(__dirname, 'public')));
```

debajo del “*app.get*” que nos devuelve “Hello World!”
3. Guardamos y ejecutamos el proyecto *HelloWorld* desde la carpeta raíz (`node index.js`).
4. Accede a “[localhost:8080/index.html](http://localhost:8080/index.html)” (o el puerto configurado).
5. Pulsamos sobre el botón.



```
GET /index.html 200 6.842 ms - 483
GET /js/jquery-3.6.4.min.js 200 2.905 ms - 89795
GET /js/wsinvocations.js 200 2.421 ms - 280
```

**Nota:** Observa en la consola que los recursos son servidos directamente por el servidor. Se podría acceder a otros ficheros en “*public*” añadiendo su ruta en la URL.

# IMPLEMENTACIÓN ALTERNATIVA

- **jQuery.Deferred**: permite manejar varias peticiones de forma más legible:

```
function getHelloAndGoodbye() {  
    $.when (   
        $.ajax({  
            type: "GET",  
            url: "http://localhost:8080/hello" }) ,  
        $.ajax({  
            type: "GET",  
            url: "http://localhost:8080/goodbye" })  
    ).done(function(helloRes, goodbyeRes) {  
        $("#resGetHello").html(helloRes[0]);  
        $("#resGetGoodbye").html(goodbyeRes[0]);  
    }).fail(function(res) {  
        alert("ERROR: " + res.statusText);  
    });  
};
```

`$.when` internamente crea objetos *Deferred* para tratar varias peticiones asíncronas y esperar a que todas se completen.

Tendríamos que crear dos nuevos MWs en el fichero *index.js*

- **Index.html**: Crea otro `button` y otra etiqueta `h3` para invocar la nueva función:

```
<input type="button" value="GET HELLO 2" onclick="getHelloAndGoodbye()" />  
<h3 id="resGetGoodbye"></h3>
```

# ÍNDICE

- Breve introducción: AJAX y jQuery
- Invocación mediante AJAX y JQuery para HelloWorld
- **Invocación servicios web REST películas**
- Conexión de servicios con página web



# ATRIBUTOS AJAX – INVOCANDO A LA API REST

- **type** (GET; POST; PUT; DELETE)

`type: "GET",`

- **url** (ruta del recurso al que queremos acceder)

`url: http://localhost:8080/movies.`

- **dataType** (tipo de dato que esperamos recibir tras la invocación)

`dataType: "json",`

- **contentType** (tipo de dato que enviamos con la invocación)

`contentType: "application/json",`

- Al enviar un objeto JSON, tenemos que *convertirlo a una cadena de texto JSON*:

`data: JSON.stringify( { "id": "020", "name": "pepe" } )`

También se usa *JSON.stringify* para tratar un objeto JSON recibido como texto:

```
success: function(data) {  
    $("#resGetHello").html(JSON.stringify(data));  
}
```

Más información de los atributos: <https://api.jquery.com/jQuery.ajax/>

# EJEMPLO 1 – GET JSON (MOVIE)

- Realiza los pasos anteriores para el ejemplo de Películas con la BBDD MongoDB.
- Añadir en “*wsinvocations.js*”:

```
function getMovie(movieId) {  
    let myUrl = "/movies/" + movieId;  
    $.ajax({  
        type: "GET",  
        dataType: "json",  
        url: myUrl,  
        success: function(data) {  
            $("#resPelicula").html(JSON.stringify(data[0]));  
        },  
        error: function(res) {  
            let mensaje = JSON.parse(res.responseText);  
            alert("ERROR: " + mensaje.msg);  
        }  
    });  
}
```

Si la petición es hacia el mismo servidor, no hace falta indicar la URL completa, solo la ruta al recurso.

Esperamos recibir un documento JSON

Podemos acceder a los atributos de la película o pasarla a texto, como aquí.

Diferentes respuestas:

- **res.status**: código de estado
- **res.statusText**: texto asociado al código de estado.
- **res.responseText**: texto o documento JSON enviado al formar la respuesta.

# EJEMPLO 2.1 – POST JSON (MOVIE)

- Añadir en *wsinvocations.js*

```
function postMovie() {  
    $.ajax({  
        type: "POST",  
        url: "/movies",  
        contentType: "application/json",  
        dataType: "text",  
        data: JSON.stringify({  
            "title": "Dunkirk",  
            "director": "Christopher Nolan",  
            "year": 2017  
        }),  
        success: function(data) {  
            $("#resPelicula").html(data);  
        },  
        error: function(res) {  
            alert("ERROR " + res.statusText);  
        }  
    });  
}
```

Estamos enviando un documento de tipo JSON

En este ejemplo, los datos de la película los estamos poniendo sobre la marcha. Pero, como ya intuyes, esto no suele ser así.

**En tu proyecto**, los datos que se incluyan dentro de la llamada a *JSON.stringify* **deben extraerse dinámicamente**, por ejemplo de *inputs*, o bien los datos se pasarán como parámetros a la función JavaScript

# EJEMPLO 2.2 – POST JSON (MOVIE)

- Añadir en *index.html*:

```
<input onclick="getMovie('ID MOVIE')" type="button" value="GET MOVIE 'The Lion King'"/>
```

```
<input onclick="postMovie()" type="button" value="POST MOVIE 'Dunkirk'"/>
```

```
<input onclick="getMovie('ID MOVIE')" type="button" value="GET MOVIE 'Dunkirk'"/>
```

## Películas Cliente

GET MOVIE 'The Lion King'

POST MOVIE 'Dunkirk'

GET MOVIE 'Dunkirk'

**{"msg":"Film created!"}**

## Películas Cliente

GET MOVIE 'The Lion King'

POST MOVIE 'Dunkirk'

GET MOVIE 'Dunkirk'

**{"\_id":"605393449aacc81b54ec2087","title":"Dunkirk","director":"Christopher Nolan","year":2017}**

**Nota:** en el botón *Get Movie 'Dunkirk'*, tendremos que pasarle el id que genera MongoDB para la película que acabamos de crear con el POST.

# EJEMPLO 3.1 – GET ALL MOVIES

- Añadir en *wsinvocations.js*:

```
function getAllMovies() {  
  let myUrl = "/movies";  
  $.ajax({  
    type: "GET",  
    dataType: "json",  
    url: myUrl,  
    success: function(data) {  
      $("#resPelicula").html(JSON.stringify(data));  
    },  
    error: function(res) {  
      console.error("ERROR:", res.status, res.statusText);  
    }  
  });  
}
```

En este caso, en lugar de mostrar un mensaje con *alert*, dejamos marcado un error en la consola, con el código de estado y el mensaje asociado al mismo.

Abajo se muestra el resultado obtenido cuando se produce un error.

✖ ▼ ERROR: 500 Internal Server Error  
error @ [wsinvocations.js:48](#)

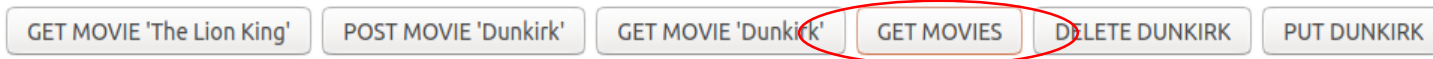
[wsinvocations.js:48](#)

# EJEMPLO 3.2 – GET ALL MOVIES

- Añadir en *index.html*:

```
<input onclick="getAllMovies()" type="button" value="GET MOVIES"/>
```

## Películas Cliente



```
[{"_id":"6050f151141c3e575f69380b","title":"The Lion King","director":["Rob Minkoff","Roger Allers"],"year":1990}, {"_id":"605393449aacc81b54ec2087","title":"Dunkirk","director":"Christopher Nolan","year":2017}]
```

- Ejercicio:
  1. ¿Cómo puedo conseguir que el *postMovie* me muestre “Film created!” quitando el “msg”? Pista: si *JSON.stringify* permite **convertir** un objeto JavaScript en una cadena en formato JSON, ¿cómo podemos **recibir** texto en formato JSON y transformarlo en un objeto JavaScript?
  2. Implementa la invocación al método **PUT** y **DELETE** de una película.
  3. Investiga **cómo iterar sobre el array de películas** que nos devuelve la API REST para mostrar las películas de forma “bonita” en el HTML.
  4. Con lo anterior, modifica la **función de éxito** de *getAllMovies()*.

# ÍNDICE

- Breve introducción: AJAX y jQuery
- Invocación mediante AJAX y JQuery para HelloWorld
- Invocación servicios web REST películas
- **Conexión de servicios con página web**

# CONECTAR SERVICIOS CON LA PÁGINA WEB

Para **invocar a los servicios** definidos desde la web, tendremos:

- Un **conjunto de campos** (para *añadir y editar* un registro).
- Una **lista/tabla** que liste los diferentes registros, así como permita trabajar con estos individualmente (*visualizar, editar y eliminar*).

Ver código de ejemplo: **listar\_videojuegos.html**  
(*simula las peticiones jugando con un array*)



**Para conseguir una conexión completa:**

- Desde los *botones*, se invocarán los servicios REST que correspondan.
- Para los servicios de tipo POST/PUT se extraerán datos de los *campos*.
- **Se controlará el *id* de forma dinámica** (en lugar de escribirlo en el código directamente como hasta ahora). De forma simple, podemos tratarlo como un dato más del registro (por ejemplo, crear un *input* donde poner el id, utilizar atributos de las propias etiquetas...).



# FORMULARIOS

Aparte de usar AJAX, se puede trabajar mediante *formularios*:

```
<form action="/people" method="POST">
  <label for="nom">Nombre:</label>
  <input type="text" name="nombre" id="nom">
  <label for="ap">Apellido:</label>
  <input type="text" name="apel" id="ap">
  <input type="submit" value="Enviar">
</form>
```

- Solo se permiten operaciones **GET** (recuperar) y **POST** (crear):
  - **GET**: Los datos se envían en la petición en forma de *query* (*req.query*):  
`/people?nombre=Pedro&apel=Delgado`
  - **POST**: Los datos se envían dentro del cuerpo de la petición (*req.body*). Es necesario incluir en la configuración:  
`app.use(express.urlencoded({extended: true}));`
- Las operaciones **PUT** y **DELETE** se pueden simular. En NodeJs, se puede hacer mediante el paquete [method-override](#):

```
<form method="POST" action="/resource?_method=DELETE">
  <button type="submit">Delete resource</button>
</form>
```

```
// override with POST having ?_method=DELETE
app.use(methodOverride('_method'))
```

# Bibliografía

## *Bibliografía:*

- 📖 **Web Development with JQuery.** York, Richard. John Wiley & Sons, Incorporated, 2015.
- 📖 **RESTful web API design with Node.js 10 : learn to create robust RESTful web services with Node.js, MongoDB, and Express.js,** Valentin Bojinov, Packt Publishing, 3ª Ed, 2018.

# Keep calm and debug

## *¡NO ARRANCA!*

- 1) ¿Has iniciado el servidor?
- 2) ¿Estás lanzando *node* desde el directorio correcto?
- 3) ¿Has recordado instalar las dependencias?
- 4) ¿Has dejado abierta la conexión en otra terminal?



## *¡¡ARRANCA PERO NO FUNCIONA!!*

- 5) ¿Has mirado los mensajes de la terminal?
- 6) ¿Se están registrando los cambios en la BBDD?
- 7) ¿Tienes MWs repetidos? Copiapega: tu peor enemigo.
- 8) ¿Has probado a poner mensajes "*console.log*"?
- 9) ¿Has tratado de acotar el problema? Comenta/descomenta.



## *¡¡¡NADA, NO FUNCIONA!!!*



- 10) Vuelve a empezar.

