**Laboratorio: Tecnologías JavaSript y Ajax**

Roberto León Grande

Docente

Dr. David Flores Vasconcelos

Desarrollo de aplicaciones en red

Universidad Internacional de La Rioja

Escuela Superior de Ingeniería y Tecnología

Grado en Ingeniería Informática

Madrid, 2021

**Resumen**

Desarrollo web utilizando las tecnologías JavaScript y Ajax. Se presenta una página inicial en la que se describe la actividad y los ejercicios. En ella se encuentran el menú para acceder a las otras dos páginas en las que se resuelven los ejercicios de JavaScript y de Ajax.

**Prefacio**

Este documento se redacta para dar solución al laboratorio “Tecnologías JavaScript y Ajax”, propuesto en la asignatura de Desarrollo de Aplicaciones en Red, perteneciente al grado de Ingeniería Informática de la Escuela Superior de Ingeniería y Tecnología de la UNIR

**Tabla de Contenidos**

[**Introducción** 1](#_Toc90398747)

[**Desarrollo** 2](#_Toc90398748)

[**Estructura general del proyecto** 2](#_Toc90398749)

[Estructura Html 2](#_Toc90398750)

[Ejercicio JavaScript 2](#_Toc90398751)

[Ejercicio Ajax 4](#_Toc90398752)

[El resultado 5](#_Toc90398753)

[**Referencias** 6](#_Toc90398754)

# **Introducción**

El objetivo principal de este laboratorio es afianzar los conocimientos en JavaScript y Ajax mediante la resolución de los siguientes ejercicios:

* JavaScript
  + Detectar si la cadena de entrada en un palíndromo.
  + Escribe un programa que pida dos números y escriba en la pantalla cuál es el mayor.
  + Escribe un programa que pida una frase y escriba las vocales que aparecen.
  + Escribe un programa que pida una frase y escriba cuántas veces aparecen cada una de las vocales.
* Ajax: A partir de la web proporcionada, se pide añadir el código necesario para:
  + Al cargar la página, el cuadro de texto debe mostrar por defecto la URL de la propia página.
  + Al pulsar el botón Mostrar Contenidos, se debe descargar mediante peticiones AJAX el contenido correspondiente a la URL introducida por el usuario. El contenido de la respuesta recibida del servidor se debe mostrar en la zona de Contenidos del archivo.
  + En la zona Estados de la petición se debe mostrar en todo momento el estado en el que se encuentra la petición (no iniciada, cargando, completada, etc.).
  + Mostrar el contenido de todas las cabeceras de la respuesta del servidor en la zona Cabeceras HTTP de la respuesta del servidor.
  + Mostrar el código y texto de la respuesta del servidor en la zona Código de estado.

# **Desarrollo**

## **Estructura general del proyecto**

El proyecto se estructura con los siguientes archivos:

* Index.html es el end point del proyecto y como tal, la página principal en la que se exponen los ejercicios.
* ejercicio\_JS.html: vista para presentar la resolución de los ejercicios propuestos de JavaScript
* ejercicio\_Ajax.html: vista para presentar la resolución del ejercicio propuesto de Ajax
* codeJS.js: código JavaScript que resuelve los ejercicios de JavaScript
* codeAjax.js: código JavaScript que resuelve el ejercicio de Ajax
* lab1\_style.css: código Css que modela las páginas html del proyecto. Este archivo es común para todos.

**El proceso de desarrollo**

### Estructura Html

En primer lugar, se maquetan las páginas html para poder presentar los ejercicios partiendo del archivo facilitado en la documentación del laboratorio. Se plantea una página inicial *index.html* en la que se muestra la descripción del laboratorio. Se maqueta una cabecera y un pie de página común para todas. En la cabecera se muestra el título y el menú para poder moverse entre los ejercicios y el enunciado. En el pie se muestra el autor y su correo . El cuerpo es específico de cada página para adaptar la presentación a las necesidades del problema. Toda la maquetación utiliza el archivo lab1\_style.css para conseguir la apariencia presentada.

### Ejercicio JavaScript

La página ejercicio\_Js.html muestra los diferentes ejercicios propuestos y mediante el archivo codeJS.js, anteriormente citado, se resuelven y presentan los resultados de manera dinámica sin necesidad de recargar toda la web utilizando el archivo .css para darle el estilo. Para ello, se crean etiquetas input´s con id y nombre. Estas serán utilizadas para captar y presentar la información.

El archivo codeJS.js contiene el código que resuelve el problema. En primer lugar, se añade un escuchador a todo el documento html para capturar cualquier evento que ocurra. Seguidamente se comprueba que los id correspondientes a los eventos existan. Si es así, se capturan y se asignan a una variable y se les añade un escuchador. El listener se añade mediante addEvenListener() y recibe como parámetros el tipo de evento que se tiene que producir y la función a la que se llama cuando se produce. Para cada ejercicio se ha creado una función independiente.

* function ejercicio\_1():

Se recoge el valor de la cadena introducida, se ponen todas las letras en minúsculas, se eliminan los espacios y se almacena en la variable *text.* Seguidamente se captura el objeto input donde se van a mostrar los resultados. Por último, se compara el string almacenado en *text* con el mismo, pero invirtiendo su orden. En función del resultado se muestra si es o no palíndromo.

* function ejericico\_2():

Se capturan los valores introducidos y se almacenan en las variables *num-1 y num-2*. Se captura el objeto *numResult* para poder mostrar el resultado mediante *innerHTML*. Por último, se comprueba que el valor es numérico y se evalúa cuál de los números es mayor, presentándolo en la web mediante su input.

* function ejercicio\_3():

Se captura el valor del input que hay cada vez que se produce un evento y se almacena en la variable *cadena.* Seguidamente se almacena el input del resultado. Para terminar, se evalúa si la cadena tiene valores y si los tiene, mediante una llamada a la función *vocales()*, se obtiene un mapa con las vocales que tiene la cadena y la cantidad de ocurrencias. Como en este ejercicio solo se necesitan las vocales, se recorre el mapa en función de sus claves, que corresponden a las letras, y se almacenan en la variable *vocalesExistentes* para posteriormente mostrarlas en el input mediante .innerHTML.

* function ejercicio\_4():

Al igual que en el caso anterior se capturan los inputs, tanto de entrada de datos como en el que se mostrará el resultado y se almacenan. Se comprueba si la variable *cadena* tiene valores y, si es así, se llama a la función *vocales()* para obtener un mapa clave valor con las vocales que aparecen y el número de ocurrencias. En este caso es necesario tanto la clave ( la letra que es) como el valor ( el número de ocurrencias), por lo que se recorre todo el mapa y se guarda las tuplas clave: valor en la variable *resultado*. Para terminar, se muestran en el input mediante .innerHTML

* function vocales():

Esta función se crea para ser utilizada por los ejercicios 3 y 4. De esta manera se consigue menor acoplamiento y reutilización de código. La función recibe la cadena a evaluar y la pasa a minúsculas mediante .toLowerCase(). Seguidamente se declara un array en el que se almacenan todas las ocurrencias de las vocales por medio de la función .match() la cual recibe una expresión regular con los elementos a seleccionar. Una vez se tienen todas las ocurrencias se seleccionan por vocal y se guardan en la variable correspondiente mediante la función .filter() que selecciona los elementos del array que coincidan con los indicados por parámetro

Por último, se declara la variable *mapVocales* de tipo mapa y se almacenan en ella las vocales que han tenido alguna ocurrencia, siendo la clave la vocal y el valor el número de ocurrencias. De esta manera se está seleccionando en tiempo de ejecución el valor de las claves de la variable. Es por esta razón por las que se elige esta estructura y no un objeto complejo.

Se retorna el mapa para que las funciones de los ejercicios hagan el uso de él que necesiten.

### Ejercicio Ajax

La página ejercicio\_Ajax.html muestra la presentación planteada. En este caso se ha modificado el código html dado y el css para poder colocar los elementos en una disposición que facilite su lectura. Se ha respetado la disposición de los inputs de entrada de la url y el botón y se han modificado el resto. Se han dispuesto el estado y el código primero y el contenido y las cabeceras debajo.

En el archivo codeAjax.js añade un escuchador al documento para capturar cualquier evento que ocurra. Se evalúa la existencia del input en el que aparece la url, se elimina el posible contenido inicial y se le asigna el valor de la url actual mediante *location.href*. Después se evalúa la existencia del id del botón, y se le añade el listener con la llamada a la función que captura el objeto Http. Seguidamente se valida si existe el formulario y se crea la variable con su objeto para validar las url´s introducidas por el usuario y se valida y crea la variable de estado para poder mostrar el estado inicial de la petición desde que se carga la página para dar mayor información al usuario.

La función *enviarObjetoHttp()* captura el objeto del recurso y evalúa su contenido. Si está vacío o la url no es válida, muestra una alerta y recarga la página. Si todo es correcto, crea las variables y captura los objetos para poder almacenar los códigos, contenidos y cabeceras, borrando su contenido inicial. Seguidamente se crea la variable objetoHttp y se utiliza:

* open(GET, variable) para abrir la petición
* .*onerror* para capturar los errores como por ejemplo de CORS.
* *.ontimeout* para evitar esperas excesivas
* *getAllTesponseHeaders* para recoger todas las cabeceras
* *onreadystatechange* para capturar los cambios de estado de la petición
* *onload* para cargar el código correspondiente cuando se termina la petición

### El resultado

La web no se encuentra alojada en ningún sitio público, por lo que se adjunta el proyecto con todos los archivos para su ejecución. Hay que recordar que para comprobar el funcionamiento del ejercicio de Ajax es necesario poder realizar peticiones por lo que se recomienda utilizar el plugin para VSCode “Live Server” de Ritwick Dey.