AJAX

DPT. INFORMÁTICA

-IES TRASSIERRA-

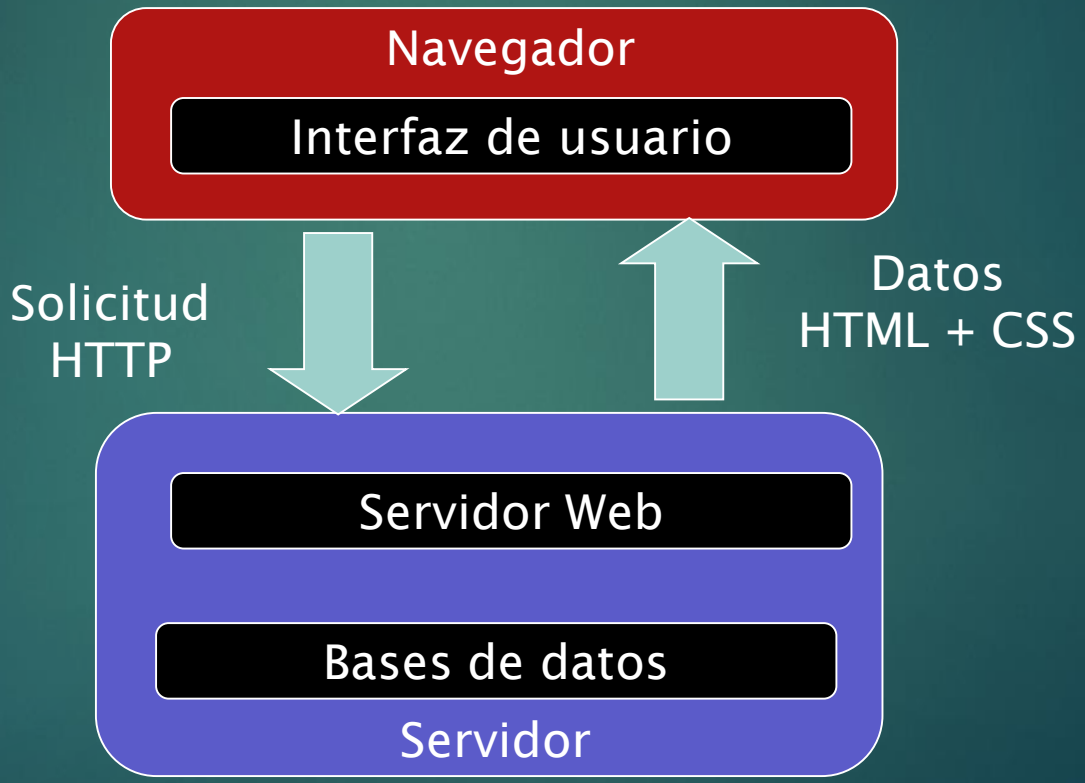
AJAX –

- AJAX proviene de Asynchronous JavaScript y XML, que se puede traducir en JavaScript asíncrono y XML.
- Basado en JavaScript y respuestas HTTP.
- AJAX es una tecnología para solicitar del servidor contenido y mostrarlo sin actualizar la página entera.
- AJAX no es un nuevo lenguaje de programación, sino una nueva forma de usar las normas existentes.
- AJAX es bueno para crear aplicaciones mejores, más rapidas y más amigables.
- Ajax está en todas partes!! Google, facebook, Gmail, Twitter, etc

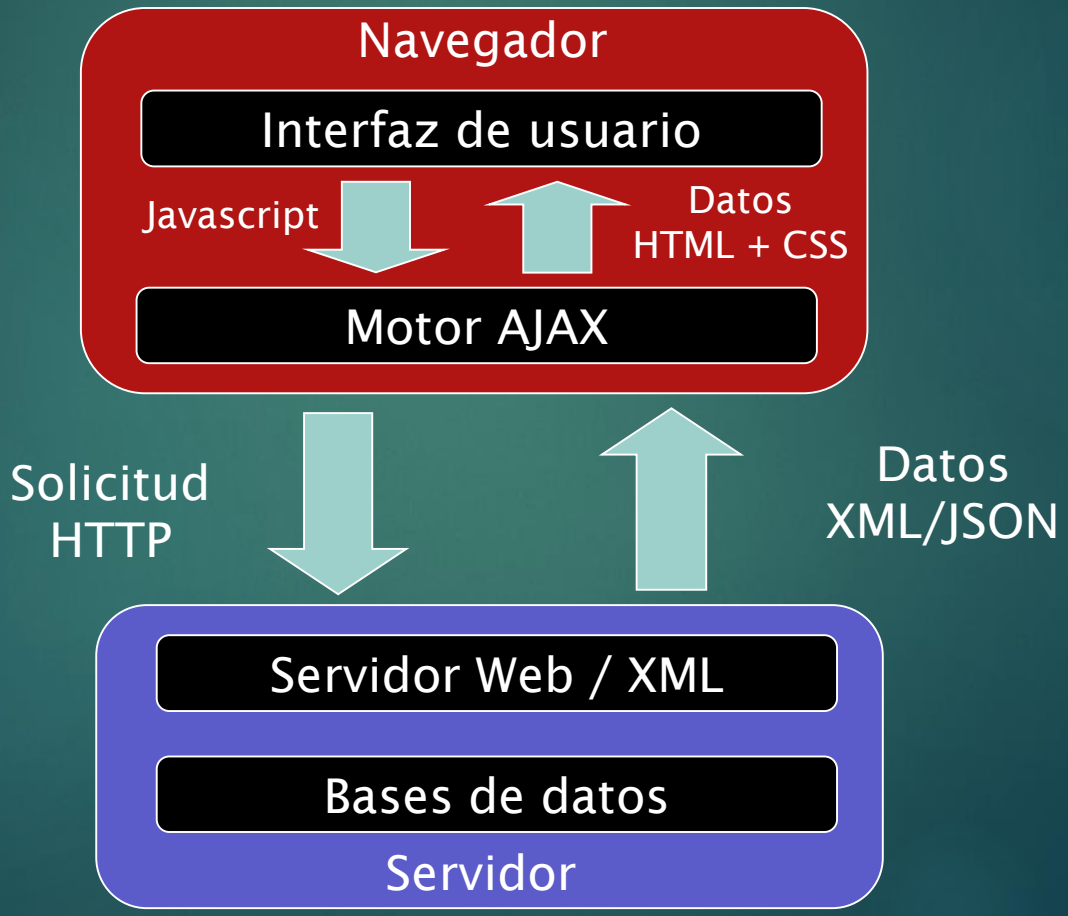
Requisitos

- HTML / XHTML
- JavaScript
- Conocimientos de XML, JSON y CSS.

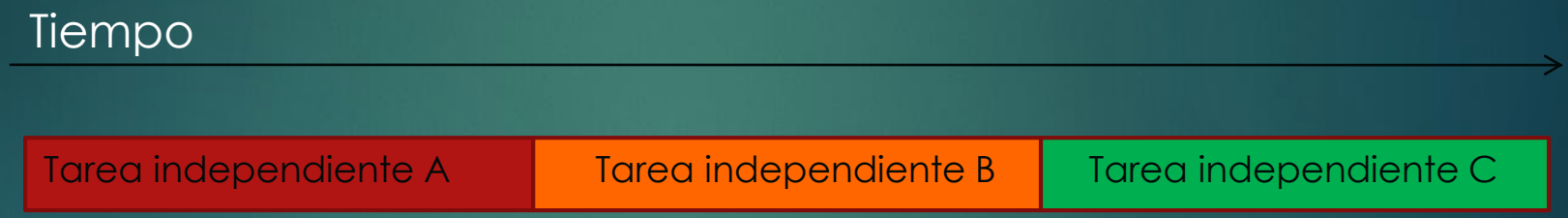
Modelo clásico de aplicaciones Web



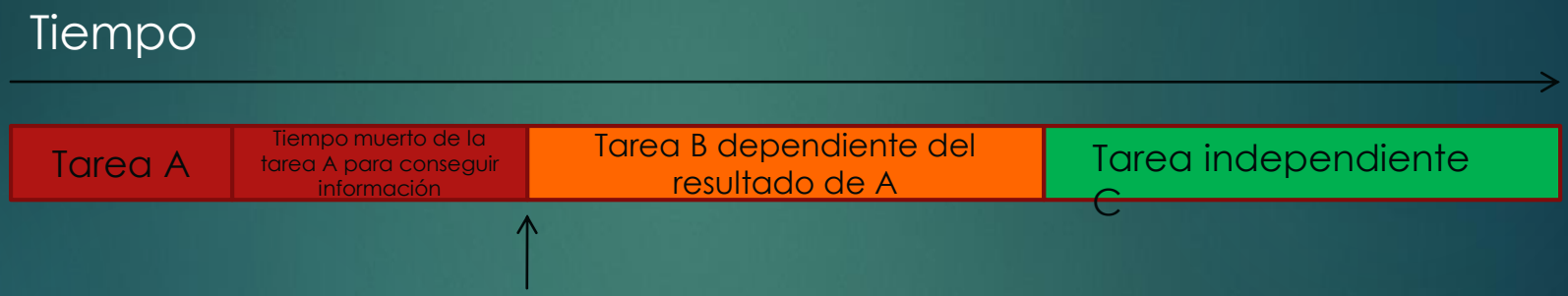
Modelo AJAX de aplicaciones Web



AJAX - Programación asíncrona - single threaded



AJAX -Programación asíncrona -



¿Dónde utilizar AJAX?

- Comunicación rápida entre usuarios
- Interacción a través de formularios
- Votaciones, encuestas, valoraciones, etc.
- Filtrado y manipulación de datos o resultados de búsqueda
- Autocompletado de campos de texto usados comúnmente

AJAX - XMLHttpRequest

- Con AJAX, JavaScript puede comunicarse directamente con el servidor, usando el objeto **XMLHttpRequest**. Con este objeto, JavaScript puede comunicarse con el servidor, sin necesidad de recargar la página.

- El objeto XMLHttpRequest nos permite la transferencia de datos en formato XML desde los script del navegador (JavaScript, JScrip, VBScript, etc.) a los del servidor (PHP, ASP, etc.) e inversamente.

AJAX - XMLHttpRequest

- La piedra angular de AJAX es el objeto XMLHttpRequest
- Según el navegador se usa diferentes métodos para crear el objeto XMLHttpRequest.
- El desarrollador no conoce el navegador que usará el usario.
- Internet Explorer usa **ActiveXObject**, mientras que otros navegadores utilizan el objeto integrado de JavaScript llamado **XMLHttpRequest**
- Para crear este objeto, y hacer frente a los diferentes navegadores, vamos a utilizar una declaración "try and catch"

AJAX - XMLHttpRequest

- Crear el objeto XMLHttpRequest.
- Comprobar navegadores.
- Si la prueba falla, entonces intentar primero el código de las nuevas versiones y después la más antigua de IE.
- Devolver un mensaje de error para los navegadores más antiguos.

AJAX – Creación XMLHttpRequest


```
function crearConexion() {
    let objeto;
    if (window.XMLHttpRequest) {
        objeto = new XMLHttpRequest();
    } else if (window.ActiveXObject) {
        try {
            objeto = new ActiveXObject("MSXML2.XMLHTTP");
        } catch (e) {
            objeto = new ActiveXObject("Microsoft.XMLHTTP");
        }
    }

    return objeto;
}
```

AJAX -XMLHttpRequest

Propiedades	Descripción
readyState	Devuelve el estado del objeto que sigue: 0=sin inicializar, 1=abierto, 2=cabeceras recibidas, 3=cargando y 4=completado
responseBody	Devuelve la respuesta como un array de bytes.
responseXML	Devuelve la respuesta en XML. Esta propiedad devuelve un objeto documento XML.
responseText	Devuelve la respuesta como una cadena
status	Devuelve el estado como un número (404 "Not Found").
statusText	Devuelve el estado como una cadena.

AJAX -XMLHttpRequest

Métodos	Descripción
abort	Cancela la petición en curso.
getAllResponseHeaders()	Devuelve el conjunto de cabeceras HTTP como una cadena.
getResponseHeader(cabera)	Devuelve el valor de la cabecera HTTP especificada.
open(método, URL[, asíncrono[,nombreUsario[, clave]]])	**Método**: "GET", "POST". Con "GET" los parámetros de la petición se codifican en la URL y con "POST" en las cabeceras de HTTP. **URL:**puede ser una URL relativa o completa. **asíncrono:** *true*, indica el que proceso del script continúa después del método *send(),* sin esperar a la respuesta. *false*, indica que el script se detiene hasta que se complete la operación
send([datos])	Envía la petición

AJAX -XMLHttpRequest

Eventos	Descripción
onreadystatechange	Evento que se dispara con cada cambio de estado
onabort	Evento que se dispara al abortar la operación
onload	Evento que se dispara al completar la carga.
onloadstart	Evento que se dispara al comenzar la carga.
onprogress	Evento que se dispara periódicamente con información de estado.

AJAX -XMLHttpRequest

Ejemplo

```
function verMensaje(){
    // crear objeto xmlHttpRequest
    let xmlHttp= crearConexion();
    //abrimos la conexión al script
    xmlHttp.open("GET",“HolaMundo.txt",true);
    //preparar la función de respuesta
    xmlHttp.onreadystatechange=function(){
        if (xmlHttp.readyState==4 && xmlHttp.status==200){
            $('#mensaje').text(xmlHttp.responseText);
        }
    }
    xmlHttp.send(); //iniciar conexión
}
```

AJAX -XMLHttpRequest

Análisis detallado

La aplicación AJAX del ejemplo anterior se compone de cuatro grandes bloques:

- ✓ instanciar el objeto XMLHttpRequest.
- ✓ Abrir la petición al servidor.
- ✓ preparar la función de respuesta.
- ✓ ejecutar la función de respuesta.

AJAX -XMLHttpRequest

Paso de parámetros con GET

Mediante GET los datos son enviados a través de la URL y no se envía nada en el método '*send()*'

Para añadir más parámetros se debe colocar el símbolo '&' al comienzo de cada nuevo dato.

```
xmlHttp.open("GET","archivo.php?valor=get");
//"GET","ejemplo3_4.php?valor=GET&nombre=LL",true
xmlHttp.send(null); // xmlHttp.send();
```

AJAX -XMLHttpRequest

Paso de parámetros con POST

Mediante POST los datos no son enviados a través de la URL, se manda por medio del método *send()*.

Debemos cambiar las cabeceras de nuestro objeto **xmlHttp** para poder enviar datos.

Para añadir más parámetros se debe colocar el símbolo '&' al comienzo de cada nuevo dato

AJAX -XMLHttpRequest

Paso de parámetros con POST

Ejemplo//

```
xmlHttp.open(“POST","archivo.php");
xmlHttp.setRequestHeader("Content-Type",
                "application/x-www-form-urlencoded");
xmlHttp.send("valor='post'"); //("valor=POST&nombre=LL")
```

AJAX con jQuery

La biblioteca jQuery tiene un conjunto completo de capacidades de AJAX que **nos permite cargar datos desde un servidor sin actualizar la página del navegador.**

Los métodos que permiten AJAX son: **$.load()**, **$.get()**, **$.post()**, **$.getJSON()**, **$.getScript()**, **$.ajax()**

AJAX con jQuery –load()

- **load()->** Inicia una solicitud Ajax a la URL especificada con parámetros opcionales.

  Sintaxis:

  $(*selector*).load(*URL,data,callback*);

AJAX con jQuery –load()-

- **Argumentos:**
  - **URL:** es la cadena que contiene la url a cargar. (obligatoria).
  - **data:** parámetros que podemos pasar para ser tratados por el destino de la url. (opcional).
  - **callback:** función invocada después de procesarse la url. (opcional). Contiene estos parámetros:
    - **responseTxt** - *contienen los datos resultantes de la solicitud*.
    - **statusTxt** - *devolverá el estado de la respuesta. "success", "error", "timeout" o "parsererror".*
    - **xhr** - devuelve el objeto XMLHttpRequest . *XHR.status o XHR.statusText*

AJAX con jQuery –load()-

- Ejemplo//

```
$("#div1").load("demo_test.txt", function(responseTxt, statusTxt,
    xhr){
        if(statusTxt == "success"){
          alert("La carga ha sido satisfactoria");
        }else if(statusTxt == "error")
          alert("Error: " + xhr.status + ": " + xhr.statusText);

        }
      });
```

AJAX con jQuery –$.get()

- $. get () -> carga los datos en el servidor mediante una petición GET HTTP.

  Sintaxis:

  $.get(*URL,data,callback*);

AJAX con jQuery –$.get()

- *funcion(responseTxt, statusTxt, XHR)-> opcional, es la función que se ejecutará si la solicitud es correcta:*
  - *responseTxt: contienen los datos resultantes de la solicitud.*
  - *statusTxt: devolverá el estado de la respuesta. "success", "error", "timeout" o "parsererror".*
  - *XHR: devuelve un objeto con datos de la respuesta del servidor. XHR.status o XHR.statusText*

AJAX con jQuery –$.get()

- Ejemplos://
  - $.get("test.php");
  - $.get("test.php", { nom:“María", ape:“Roca”});
  - $.get("test.php", {nom:“María“,ape:“Roca”},

```
function(responseTxt, statusTxt, xhr){
    if(statusTxt == "success"){
      alert(“La carga ha sido satisfactoria");
    }else if(statusTxt == "error")
      alert("Error: " + xhr.status + ": " + xhr.statusText);

    }
  })
```

AJAX con jQuery –$.post()

- $. post () -> carga los datos en el servidor mediante una petición POST HTTP.

  Todo igual que el método $.get().

AJAX con jQuery – $.getJSON()

- $. getJSON () -> carga los datos en el servidor mediante una petición GEt HTTP.

  Todo igual que el método $.get().

AJAX con jQuery –$.ajax()

- $. ajax () -> Este método abarca los métodos anteriores. Es preferible utilizar este método, ya que ofrece más características y su configuración es muy comprensible.
- Sintaxis:

```
$.ajax(opciones);
```

Opción	Descripción
async	Indica si la petición es asíncrona. Por defecto es true
beforeSend	Permite indicar una función que modifica el objeto XML antes de realizar la petición.
complete	Permite establecer la función que se ejecuta cuando una petición se ha completado (desuso)

AJAX con jQuery –$.ajax()

Opción	Descripción
data	Información que se incluye en la petición. Se utiliza para enviar parámetros al servidor.
datatype	Tipo de dato: xml, html, script, json
error	Indica la función que se ejecuta cuando se produce un error. (desuso)
success	Permite establecer la función que se ejecuta cuando una petición se ha completado de forma correcta. La función recibe los datos solicitados en el formato indicado (desuso)
timeout	Indica el tiempo máximo, en milisegundos, que la petición espera la respuesta del servidor antes de anularlo.
type	Se indica POST o GET.
url	La URL del servidor a la que se realiza la petición

AJAX con jQuery –$.ajax()

- Ejemplos//

```
$.ajax({
    url: '/ruta/hasta/pagina.php',
    type: 'POST',
    async: true,
    data:{parametro1:valor1,parametro2:valor2},
    success: procesaRespuesta,
    error: muestraError
});
```

AJAX con jQuery –$.ajax()

- Ejemplos//Últimas versiones con promesas

```
$.ajax({
    url: '/ruta/hasta/pagina.php',
    type: 'POST',
    async: true,
    data:{parametro1:valor1,parametro2:valor2},
    });
    .done(function(response, textStatus, jqXHRs){
      instrucciones…
    })
    .fail(function(response, textStatus, errorThrown){
      instrucciones…
    });
    .always(function() {
        console.log( "completado" );
    });
```

Programación AJAX ES6

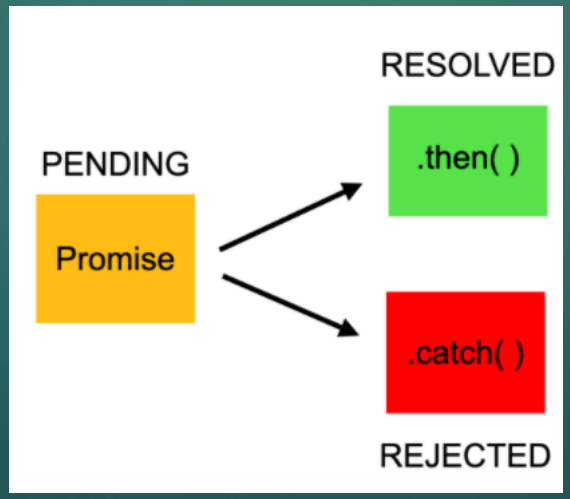
Una promesa, como concepto, es un objeto que nos va a devolver un resultado cuando una operación haya finalizado.

Las promesas intentan resolver el problema de asincronía de una forma mucho más elegante y práctica que, por ejemplo, utilizando funciones **callbacks**

Programación AJAX ES6

Estados de un objeto Promesa

- **Pendiente**: estado inicial, antes de que la promesa tenga éxito o falle.
- **Resuelto (Resolved):** Promesa completada.
- **Rechazado (Rejected)**: Promesa fallida.



Programación AJAX ES6

Creación de una promesa

En primer lugar, se usará un constructor para crear un objeto Promesa

```
const myPromise = new Promise();
```

Se necesitan dos parámetros, uno para el éxito (resolver) y otro para el error (rechazar).

```
const myPromise = new Promise((resolve, reject)=>{
    //condition
});
```

Programación AJAX ES6

Creación de una promesa

Finalmente, habrá una condición. Si se cumple la condición, la Promesa se resolverá, de lo contrario será rechazada

```
const myPromise = new Promise((resolve, reject)=>{
    let condición;
    if (condición == 20){
        resolve ("Promesa ha sido resuelta satisfactoriamente");
    } else {
        reject ("Promesa rechazada");
    }
});
```

Programación AJAX ES6

Uso de una promesa

- **then()** para promesas resueltas
- **catch()** para promesas rechazadas

```
myPromise
.then ( message =>{
        console.log(message);
})
.catch ( message =>{
         console.log(message);
});
```

AJAX -Fetch

Con la última versión ES6, ha llegado nuevas formas de realizar peticiones o llamadas en AJAX. Ya no será necesario usar JQuery o un framework que nos simplifique el trabajo. La nueva API Fetch es una API de Javascript basada en promesas para realizar solicitudes HTTP asíncronas en el navegador de forma similar a xmlHttpRequest.

AJAX -Fetch

- Get (leer datos)

```
fetch('https://httpbin.org/ip')
    .then(function(response) {
        return response.text();
    })
    .then (function(data) {
        console.log('data = ', data);
    })
    .catch(function(err) {
        console.error(err);
    });
```

AJAX -Fetch

- Post (crear nuevos datos, actualizar y borrar)

```
fetch('http://ejemplo.com/api/user', {
    method: 'POST',
    body:  datos
 })
.then(function(response) {
        return response.json(); // return response.text
    })
.then(function(data) {
        console.log('data = ', data);
    })
.catch(function(err) {
        console.error(err);
    });
```

AJAX -Fetch

- datos

  Los datos podrán pasarse mediante objetos **formData**:

  ```
  let datos= new formData();
  ```

  Añadir parámetros:

  ```
  datos.append("chip", ”111A”);
  datos.append(“nombre”, “ganso”);
  ```

AJAX - AXIOS

Axios es una librería Open Source que permite realizar operaciones como cliente Http, basado en el modelo de las promesas.

Axios destaca por su sencillez y ligereza, lo cual lo hace muy indicado para proyectos. También proporciona una capa unificada para la realización de peticiones AJAX, independiente de otros framework que usemos en el proyecto.

AJAX – AXIOS- Instalación

Axios puede instalarse mediante el gestor de paquetes NPM, copiando la librería de forma local o desde un CDN.

```
<script src="https://unpkg.com/axios/dist/axios.min.js"></script>
```

AJAX – AXIOS- Get Request


```
axios.get('/user', {
 params:{
        ID:12345
})
 .then(function (response) {
 console.log(response);
 })
 .catch(function (error) {
console.log(error);
 });
```

AJAX – AXIOS- Post Request


```
axios.post('/user', {
        ID:12345,
        firstName: 'Fred',
        lastName: 'Flintstone'
 })
 .then(function (response) {
 console.log(response);
 })
 .catch(function (error) {
console.log(error);
 });
```

AJAX – AXIOS- Put Request


```
axios.put('/user/12345', {
        firstName: 'Paul',
        lastName: 'Red'
 })
 .then(function (response) {
 console.log(response);
 })
 .catch(function (error) {
console.log(error);
 });
```

AJAX – AXIOS- delete Request


```
axios.delete('/user', {
        ID:12345'
 })
 .then(function (response) {
 console.log(response);
 })
 .catch(function (error) {
console.log(error);
 });
```