**AJAX**

Acrônomo de **A**synchronous **J**avaScript **A**nd **X**ML;

**JavaScript** e **XML** como assíncronos;

**Técnica de programação**, utiliza Javascript e que possibilita a **comunicação** assíncrona entre **front-end e back-end** de aplicações Web;

Inicialmente **disponibilizado na versão 4.0** do Internet Explorer;

Potencializou a criação do conceito de web 2.0;

Atualmente o formato mais utilizado para comunicação entre front e back, é o JSON.

**Modelo conceitual de uma aplicação Web**

Cliente, front-end onde browser é executado -> back-end, servidor onde roda a aplicação que geralmente guarda regras de negócio e o banco de dados;

Entre eles, existe o meio físico de comunicação, a internet.

XMLHTTP Request – cliente realiza requisição para o servidor. Gerencia processo de envio da requisição e captura da resposta. Durante esse período, aplicação web não trava.

**Ajax – O que são requisições síncronas e assíncronas**

Aplicações web – inseridas: cliente/servidor;

**Síncrono** – quando efetuamos uma requisição, o browser responsável por ela ficará aguardando até que uma resposta seja enviada pelo servidor;

Cada requisição deve ser finalizada para possibilitar que uma nova requisição seja efetuada.

**Assíncrono** – browser é capaz de disparar diversas requisições, sem ser necessário um sincronismo entre elas;

É possível efetuar diversas requisições simultaneamente e cada uma delas será tratada de forma paralela.

**Requisições síncronas**

Cliente (**Browser**) ---**Request**---------🡪 **Servidor** Apache (htdocs)

Cliente (**Browser**) <----------**Response**-- **Servidor** Apache (htdocs)

Enquanto requisição está sendo feito, fica pendente para o usuário.

**Requisições assíncronas – 1**

**Efetuando requisições HTTP via XMLHttpRequest**

Ao invés de navegar entre páginas distintas, haverá apenas **uma página requisitando** **conteúdo dinâmico**;

Página deve sofrer **request a cada solicitação de dados**.

**function requisitarPagina(url) {**

**let ajax = new XMLHttpRequest();**

**ajax.open("GET", url);**

**//responsável por configurar url requisitada, podemos configurar também, método utilizado nesse processo**

**ajax.send();**

**Requisições assíncronas – 2**

**Estados de requisição**

ReadyState

- **0**: request not initialized (**requisição não iniciada**);

- **1**: server connection established (**conexão estabelecida com servidor**);

- **2**: request received (**requisição recebida**);

- **3**: processing request (**processando requisição**);

- **4**: request finished and response is ready (**requisição finalizada**).

**function requisitarPagina(url) {**

**let ajax = new XMLHttpRequest();**

**//requisição não iniciada, state = 0**

**console.log(ajax.readyState);**

**ajax.open("GET", url);**

**//responsável por configurar url requisitada, podemos configurar também, método utilizado nesse processo**

**//conexão estabelecida com servidor, state = 1**

**console.log(ajax.readyState);**

**//lógica que olhe progresso da requisição**

**ajax.onreadystatechange = () => {**

**console.log(ajax.readyState);**

**}; //executada pelo próprio ajax**

**ajax.send();**

**//na prática, geralmente só utilizamos o estado 4, pois ele indica que a requisição foi finalizada e com isso podemos recuperar a resposta fornecida pelo servidor e tomar alguma ação**

**}**

**Requisições assíncronas – 3**

**Aplicando um loading**

**1** – No disparo da requisição apresentar o gif de loading;

**2** – Ao atingir o estado de 4 a requisição remover o gif de loading.

//incluir gif de loading na página

        if (!document.getElementById("loading")) {

          //se não haver uma imagem 'loading', incluir

          let imgLoading = document.createElement("img");

          imgLoading.id = "loading";

          imgLoading.src = "loading.gif";

          imgLoading.className = "rounded mx-auto d-block";

//lógica que olhe progresso da requisição

        ajax.onreadystatechange = () => {

          if (ajax.readyState == 4) {

            console.log("requisição finalizada com sucesso!");

            document.getElementById("loading").remove(); //removendo imagem quando requisição termina de ser executada

          }

        }; //executada pelo próprio ajax

**Requisições assíncronas – 4**

**Status da requisição**

**Status no protocolo HTTP:**

- Respostas informativas (100-199);

- Respostas de sucesso (200-299);

- Mensagens de redirecionamento (300-399);

- Erros do lado do cliente (400-499);

- Erros do lado do servidor (500-599).

**5 grupos distintos com diversos status possíveis.**

Status mais utilizados no dia a dia:

**200 OK – Sucesso;**

**404 Not Found – Não encontrado.**

        document.getElementById("conteudo").innerHTML = "";

Limpando a tela

 if (ajax.readyState == 4 && ajax.status == 200) {

            document.getElementById("conteudo").innerHTML =

              "requisição finalizada com sucesso, o status é 200";

            //document.getElementById("loading").remove(); //removendo imagem quando requisição termina de ser executada

          }

          if (ajax.readyState == 4 && ajax.status == 404) {

            document.getElementById("conteudo").innerHTML =

              "requisição finalizada, porém o recurso solicitado não foi encontrado, o status é 404";

            // document.getElementById("loading").remove(); //removendo imagem quando requisição termina de ser executada

          }

        }; //executada pelo próprio ajax

**Requisições assíncronas – 5**

**ResponseText**

A partir da resposta de uma requisição interceptar o seu conteúdo e trabalhar esse conteúdo no front-end da aplicação.

if (ajax.readyState == 4 && ajax.status == 200) {

            document.getElementById("conteudo").innerHTML = **ajax.responseText;**

**XML – 1**

**O que é?**

XML significa e**X**tensible **M**arkup **L**anguage.

- Como podemos utilizar o XML para estabelecer a comunicação entre o front e o back-end da aplicação.

É uma linguagem de marcação – serve para estruturar e descrever informações;

Recursos para comunicação entre sistemas distintos;

Serve para troca de informações em qualquer tipo de sistema;

Compatível com diversas linguagens, pode ser gerado por diversas linguagens.

**Recuperando arquivo XML do servidor via HTTP**

      function getFilmes() {

        let XMLHttp = new XMLHttpRequesT(); //instancia atribuída a variável

        XMLHttp.open("GET", "http://localhost/ajax/filmes.html"); //método e recurso a ser recuperado através do get(url)

        XMLHttp.onreadystatechange = () => {//controla estados da requisição

        XMLHttp.send();//dispara request

      }

Onreadystatechange – evento disparado, sempre que o objeto readyState

(no momento associado a variável muda).

**if (XMLHttp.readyState == 4) {//se estado da req é igual a 4(realizada com sucesso)**

**XMLHttp.onreadystatechange = () => {**

**//controla estados da requisição**

**if (XMLHttp.readyState == 4 && XMLHttp.status == 200) {**

**//se estado da req é igual a 4(realizada com sucesso) e se status é igual a 200(OK)**

**let XMLFilmes = XMLHttp.responseText; //retorno para requisição que foi feita em filmes.xml**

**}**

**if (XMLHttp.readyState == 4 && XMLHttp.status == 404) {**

**Convertendo XML para Json**

let XMLFilmes = XMLHttp.responseText; //retorno para requisição que foi feita em filmes.xml

            let parser = new DOMParser();

            domFilmes = parser.parseFromString(XMLFilmes, "text/xml"); //método espera - string que será feita o parse + notação utilizada para a composição dessa string

            //transformando em objeto DOM

            jsonFilmes = xmlToJson(domFilmes)//transformando em obj json

Deve-se fazer isso pois o arquivo vem como texto e não tem como manipulá-lo.

**Listando os filmes da página**

Para este fim, iremos manipular o objeto JSON;

Acessa arquivo JSON, resgata filme e traz para página.

Laço de repetição e variável com caminho de acesso ao filme (questão de hierarquia)

for (let i in jsonFilmes["filmes"]["filme"]) {

              let item = jsonFilmes["filmes"]["filme"][i]; //i --- acessar índice do filme //// acessando nível de herança necessária para acessar elemento

Row e Col da lista

let divRow = document.createElement("div");

              divRow.className = "row";

              let divCol = document.createElement("div");

              divCol.className = "col";

Criando variáveis do parágrafo de cada elemento do filme utilizando a variável item para ter acesso aos dados

let p1 = document.createElement("p");

              p1.innerHTML =

                "<strong>Título: </strong> " + item["titulo"]["#text"];

let p2 = document.createElement("p");

              p2.innerHTML =

                "<strong>Resumo: </strong> " + item["resumo"]["#text"];

Criando parágrafo de gênero, que é um elemento array

let genero = ""; //texto em branco

              for (let g in item.genero) {

                //genero é array, aqui, estamos acessando cada um dos elementos desse array

                if (genero) genero += ", "; //ajustando exibição dos generos

                genero += item.genero[g]["#text"];

              }

              let p3 = document.createElement("p");

              p3.innerHTML = "<strong>Gênero: </strong> " + genero;

Criando parágrafo de elenco, que é um elemento array. Descendo níveis de hierarquia

let elenco = ""; //texto em branco

              for (let e in item.elenco.ator) {

                //genero é array, aqui, estamos acessando cada um dos elementos desse array

                if (elenco) {

                  elenco += ", ";

                } //ajustando exibição dos generos

                 elenco += item.elenco.ator[e]["#text"]; //descendo mais um nível de hierarquia para elenco retornar única pessoa

              }

let p4 = document.createElement("p");

              p4.innerHTML = "<strong>Elenco: </strong> " + elenco;

Criando parágrafo da data de lançamento. Que é exibida da seguinte forma:

(país entre parênteses)

let p5 = document.createElement("p");

              p5.innerHTML =

                "<strong>Data de lançamento: </strong> " +

                item.dataLancamento["#text"] +

                "(" +

                item.dataLancamento["@attributes"]["pais"] +

                ")";

Criando estrutura da lista e adicionando ao container pelo ID(‘listas’)

//criando estrutura com elementos

              divRow.appendChild(divCol); //colocando divCol dentro da divRow

              divCol.appendChild(p1);

              divCol.appendChild(p2);

              divCol.appendChild(p3);

              divCol.appendChild(p4);

              divCol.appendChild(p5);

              divCol.appendChild(hr); //adicionando elementos dentro de divCol

              document.getElementById("lista").appendChild(divRow); //adicionando div que contém todo o conteúdo ao DOM pelo ID

**Refactoring da comunicação com o back-end utilizando JSON**

**XML** – Estrutura informações para comunicação entre aplicações distintas;

Não é nativamente suportado como um tipo de dado dentro do front-end de aplicações web (JavaScript);

É necessário ser feita uma conversão de tipos, transformando a notação em um objeto, para fazer a manipulação das informações afetando os elementos HTML da página.

Será feita uma **conversão de um XML para JSON**;

Demonstrar como podemos **utilizar notação JSON para comunicação entre aplicações distintas**;

**Benefício**: **JSON é nativamente suportado por aplicações Web Front-End (JavaScript);**

Fica mais fácil utilizar a notação JSON do que a XML, por isso as **APIs RESTful** (baseada em JSON) vem sendo bem **mais** **utilizadas** do **que** **SOAP** (baseada em XML).

**App Pesquisa Endereço**

Pesquisa de endereço com base em um CEP;

Recurso comumente utilizado em formulários de cadastro.

Será **digitado um CEP** e com base nesse CEP **será feita uma pesquisa** em uma **API na internet**, onde será possível **recuperar** os **dados** **de endereço** desse CEP.