Session 12.
Asynchronous web applications: AJAX

Programació Multiplataforma i Distribuïda

Grau d'informàtica. EPSEVG

Desembre 2023

Jordi Esteve jesteve@cs.upc.edu

AJAX

AJAX (Asynchronous JavaScript + XML") is

a set of web **development techniques**

using many **web technologies** on the client side

to create **asynchronous web applications**.

With Ajax, web applications can send and retrieve data from a server asynchronously (in the background) without interfering with the display and behavior of the existing page.

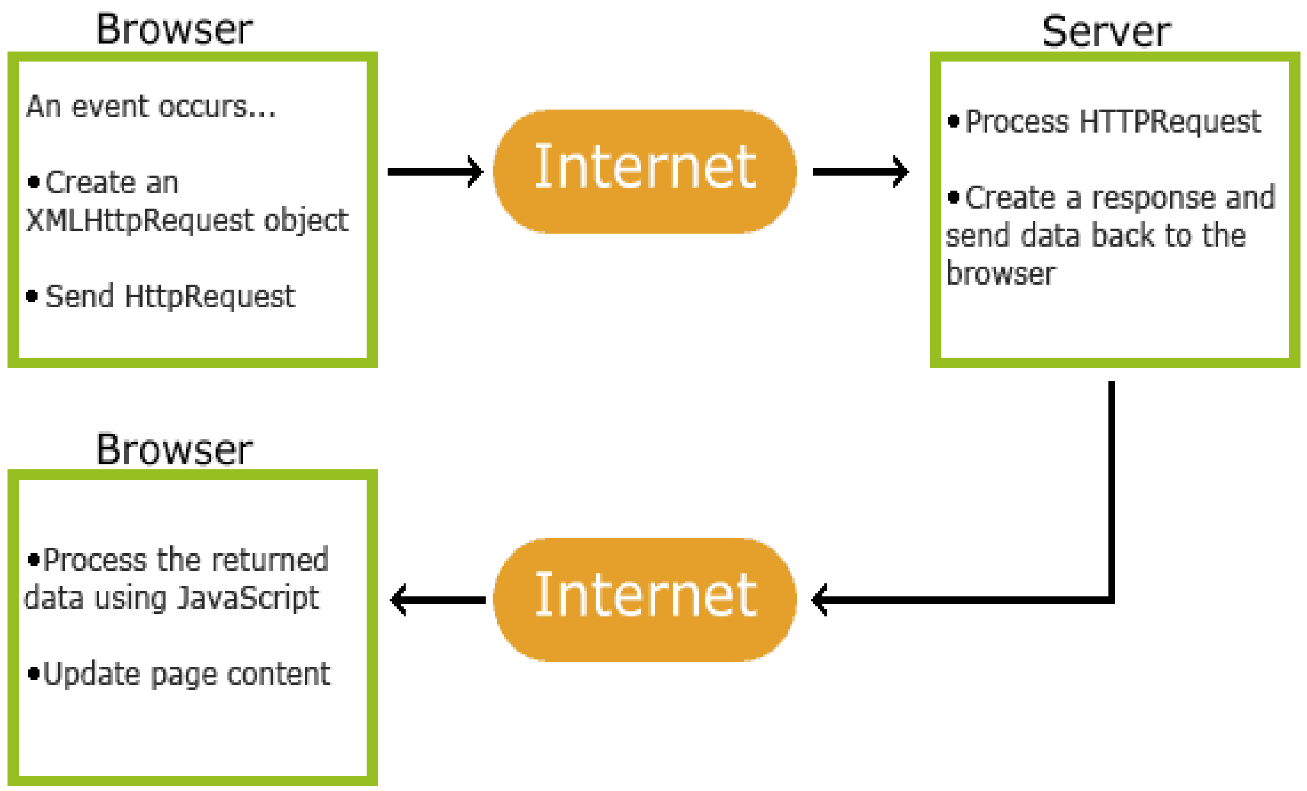
By decoupling the data interchange layer from the presentation layer, Ajax allows web pages/applications to change content dynamically without the need to reload the entire page.

Modern implementations commonly use JSON instead of XML.

AJAX: Web service client

AJAX normally uses a combination of:

- A browser built-in XMLHttpRequest object (to request data from a web server)
- JavaScript and HTML DOM (to display or use the data)


Browser
An event occurs...
•Create an XMLHttpRequest object
•Send HttpRequest
Internet
Server
•Process HTTPRequest
•Create a response and send data back to the browser
Browser
•Process the returned data using JavaScript
•Update page content
Internet

Source: www.w3schools.com

AJAX: Example of a Web service client

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
      <script>
          function loadNumber() {
            var xhttp = new XMLHttpRequest();
            xhttp.onreadystatechange = function() {
              if (this.readyState == 4) {
                  message = JSON.parse(this.responseText).message;
                  document.getElementById("number").innerHTML = message;
              }
            };
            xhttp.open("GET", "http://ubiwan.epsevg.upc.edu:3000/tasks/count", true);
            xhttp.send();
          }
      </script>
</head>
<body>
  <button type="button" onclick="loadNumber()">Get number of tasks</button>
  <p>The number of tasks are <strong id="number"></strong></p>
</body>
</html>
```

AJAX: Example of a Web service client

1. Create a *task_count_AJAX.html* file and copy the previous code.
2. Load this file in your favorite Web browser and test it.
3. Start your own task list webservice server listening at port 8000.
4. Change the URL to http://localhost:8000 and test it.
5. If it doesn't work, in the server add a route '/tasks/count' with GET verb that returns the number of tasks in the list. Restart the server and try again.

jQuery library provides us a suite of AJAX functions to simplify the code. The most used on is jQuery.ajax() (or its shortcut $.ajax()) that performs an asynchronous HTTP request. With the *dataType: "json"* option the body response is automatically parsed to convert it to a JSON object.

1. Create a *task_count_AJAX_jQuery.html* file and copy the next code.
2. Load this file in your favorite Web browser and test it.

AJAX: Example of a Web service client using jQuery

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
        <script>
                function loadNumber() {
                        $.ajax({
                                dataType: "json",
                                url: 'http://ubiwan.epsevg.upc.edu:3000/tasks/count',
                        })
                        .then(r => {document.getElementById("number").innerHTML = r.message;});
                }
        </script>
</head>
<body>
  <button type="button" onclick="loadNumber()">Get number of tasks</button>
  <p>The number of tasks are <strong id="number"></strong></p>
  <script src="https://code.jquery.com/jquery-3.4.1.min.js"></script>
</body>
</html>
```

AJAX: Web service client for task lists

1. Download the Web Service client from Atenea that manages task lists using Web services.
2. Look at the code and try to understand it. At the end of *task_view_controller.js* file you'll see that:
    - The data of the first task list is obtained from http://localhost:8000/tasks (you must start a Web service server for task lists listening at port 8000).
    - The data of the second task list is obtained from http://ubiwan.epsevg.upc.edu:3000/tasks (the Web service server of the professor, later you could change it to your own server deployed in ubiwan).
3. Load *task.html* file in your favorite Web browser and test it works right.
4. Create, update, delete, reset and switch do not work. Add the missing code in their respective ...*Controller* functions. You can look inside *listController* function how the Web service calls are done to get tasks and the count of tasks.

AJAX: Web service client for task lists

Look at this code snippet that does a web service call with jQuery ajax function to get the number (count) of tasks that matches the *where* condition:

```
$.ajax({
    dataType: "json",
    url: this.url + '/count',
    data: {params: JSON.stringify([where])}
  })
```

Notice we can send parameters in the URL if we add a *data* option. The list of parameters should be converted to a string with JSON.stringify(). By default jQuery ajax function performs GET calls, but we could use other HTTP verbs with the *method* option. The professor Web service server offers these resources:

```
app.get(['/tasks/count'], countController);     app.put('/tasks/reset', resetController);
app.get(['/', '/tasks'], getAllController);     app.put('/tasks/:id', updateController);
app.get('/tasks/:id', getController);           app.patch('/tasks/:id/switch', switchController);
app.post('/tasks', createController);           app.delete('/tasks/:id', deleteController);
```

Fixing our Web service server for task lists

1. If the first task list does not work, your Web service server has not configured all the routes. Add in your Web service server all the missing resources (see previous table); you must add the routes and the code inside the Controller functions called by the routes to compute the appropriate response.

2. Restart your Web service server for task lists listening at port 8000 and test the client now works well with the first list.

Notice that:

- The GET method receives the data in the URL query, so you can find the data in *req.query*. Some data was JSON.stringify by the client, so in the server you must do the JSON.parse to recover the original data.
- The POST, PUT, PATCH methods receive the data in the body of the request, so you can find the data in *req.body*
- To convert a list of parameters to the arguments of a function use the spread (…) operator (ES6) or the apply() function (previous ES6).

Uploading Web service server and client to ubiwan

1. Copy the files of your task list Web service **server** to the *ubiwan.epsevg.upc.edu*. You can put it outside of your *public_html* folder.
2. Change the 8000 port to your personal port (between 3001 and 3019, see your personal port in Users and Webs of PMUD students).
3. Start the Web Service server (*nohup nodejs task_API_REST_server.js &*).
4. Copy the files of your task list Web service **client** to the *ubiwan.epsevg.upc.edu* server. Put them inside of your *public_html* folder. You must modify the *task_view_controller.js* file to change the URL of the second task list from http://ubiwan.epsevg.upc.edu:3000/tasks to the URL of your Web service server.
5. Test the task list client works well. Remember, the first task list is obtained from http://localhost:8000/tasks (you must start a Web service server for task lists listening at port 8000 in your PC). And the second task list is obtained from your Web service server in ubiwan you have started in point 3.

In Users and Webs of PMUD students you can find the URL for clients and servers of each PMUD student.