Practical work : AutoCompletion system

**Evaluated skills :**

*→ Javascript events*

*→ PHP*

***Introduction :***

It is high time to practice a good deal of your knowledge. The practical work of this part will be devoted to the creation of a system of auto-completion which will be able to fetch, from a file, the cities of France starting with the letters which you will have begun to write in the field of research . The goal is to select and validate the entry of your keywords.

Unfortunately, this lab will only use AJAX through the XMLHttpRequest object to stay within reasonable dimensions. However, this is by far the most popular method today, with the use of iframes and DSL being reserved for more specific cases.

**Presentation of the exercise :**

Technologies to use

Before we start, we need to determine what kind of technology we need, because here we're not just using JavaScript, we're going to have to use other languages.

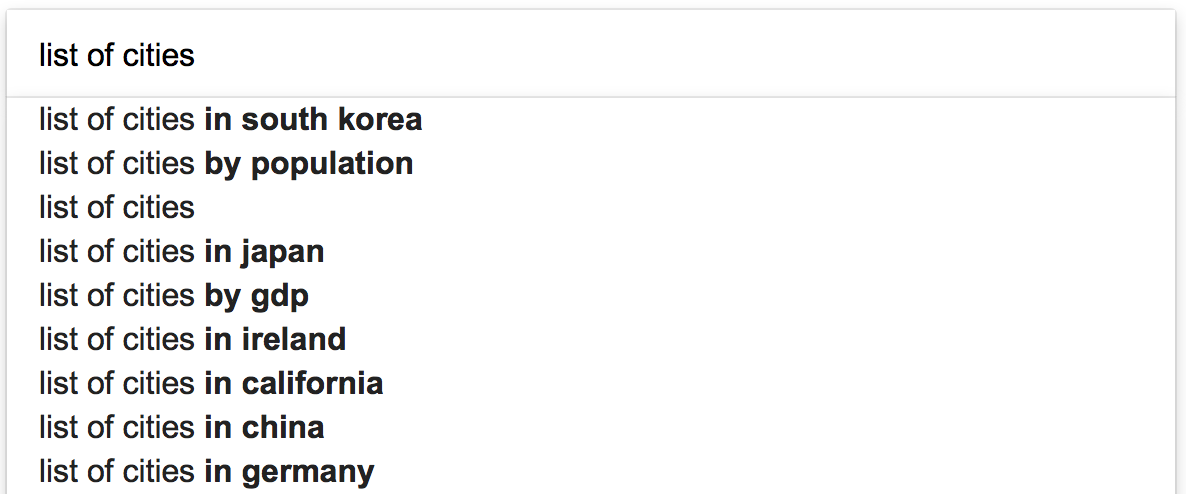
In a general framework, a system of auto-completion uses three different technologies:

* A client language with the ability to interact with a server;
* A server language capable of providing the data to the client;
* A database that stores all the data.

In our case, we will use JavaScript and PHP (although any other server language is able to do its job). We will, however, make a small sprain in the third point using a storage file rather than a database, for a very simple reason: simplify our code, especially that we do not need a base data for the few data to be saved.

Principle of auto-completion

An auto-completion system is as follows:



The principle is simple but effective: as soon as a user types a character into the field, a search is immediately performed and returned to the browser. The latter then displays the results in a small frame generally located under the field of research. The displayed results can then be browsed, either by means of the arrow keys of the keyboard (up and down), or by means of the cursor of the mouse. If we choose one of the listed results, it is automatically written in the search field instead of what was written by the user. All that remains is to start the search.

The advantage of this type of script is that you gain a lot of time: the search can be done by typing only a few characters. This is also very useful when you only know part of the search term or when you make a typo.

Design

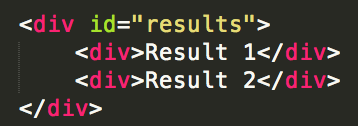
We know the principle of auto-completion and the necessary technologies. However, this does not explain how all of this should be used. We will therefore guide you to allow you to get started without too much apprehension in this vast project.

The interface

Let's start with the interface! What are we going to need? Since auto-completion is affiliated, generally, to everything in the field of research, we will need a text field to write the keywords. However, it will cause us a problem, because the browser usually saves what was written in the text fields in order to re-submit it later in the form of auto-completion, which will duplicate our system ... Fortunately , it is possible to disable this auto-completion by using the autocomplete attribute in this way:



To that we will have to add an element able to encompass research suggestions. This one will consist of a <div> tag containing as many <div> as results, like this:



Each result in the suggestions will have to change its aspect when this one will be hovered or selected by means of the arrow keys.

Regarding a possible submit button, we will do without it, because our goal is not to launch the search, but only to display an auto-completion.

For your information, since you will have to manage the arrow keys, here are the respective values ​​of the keyCode property for the Up, Down and Enter keys: 38, 40 and 13.

Client / server communication

On the contrary to what one might think, this is not a very complicated part because, basically, what are we going to do? Simply make a query to each character written to suggest a list of suggestions. We will need a function related to the keyup event of our search field, which will be responsible for making a new query to each character typed.

However, suppose we type two characters in the search field, the first query responds in 100 ms and the second in 65 ms: we will then get the results of the first query after those of the second and thus display suggestions that will no longer agree with the characters typed in the search field. The solution to this is simple: use the abort () method on the previous query if it is not completed. Thus, it is not likely to return information subsequently exceeded.

Processing and returning data

On the server side, we are going to do a basic research script on which we will not dwell too much, PHP being not our priority. The principle is to search for cities that match the letters entered in the search field. We have designed a small ZIP archive in which you will find a PHP linearized table containing the largest cities in France, you will only have to analyze it.

Link to the file

Linearizing a variable in PHP allows you to save data as a string. This is useful when you want to save a table to a file, that's what we did for cities. The functions to do this are called *serialize()* and *unserialize()*.

PHP is not necessarily your strong point (after all, you are just at the beginning of PHP), we will try to detail what you need to do to succeed in doing your research.

First of all, you need to recover the data contained in the file towns.txt (available in the archive provided above). For that, you will have to read this file with the *file\_get\_contents()* function, then to convert its contents as PHP array thanks to the function *unserialize()*.

Once done, the resulting table must be searched for results consistent with the characters typed by the user in the search field. For that, you will need a loop as well as the function *count()* to obtain the number of elements contained in the table.

To check if an index of the table corresponds to your search, you will have to use the function *stripos()*, which makes it possible to check if a character string contains certain characters, and this without regard of the case. If you find a result consistent with the search, then add it to an array (that you have previously created) through the *array\_push()* function.

Once the table has been browsed, all you have to do is sort the results with the *sort()* function, then return the data to the client ...

*Yes, but in what form? XML? JSON?*

Neither one nor the other! Simply plain text!

Why ?

For one simple reason: XML and JSON are useful for returning data that needs to be structured. If we had needed to send back, in addition to city names, the number of French habitants, businesses and administrations, then we could have considered the use of XML or JSON in order to structure all that. But in our case this is not useful because we are only returning the name of each city found.

So how to send all this back as plain text?

We could make a line break between each returned city, but it's not particularly practical to parse for JavaScript. So we will have to choose a separator character that is never used in the name of a city. In this lab, we will use the **vertical bar |** , which should allow us to return plain text in this form:

Just like *join()* in JavaScript, there is a PHP function that allows you to concatenate all the values of an array in a string with one or more separator characters: this is the *implode()* function. Once the function is used, all you have to do is return it to the client with a good old echo and analyze it on the JavaScript side.

The example here being relatively simple, we allow ourselves to dispense with data structuring. However, normally you will often use JSON.

It's yours !

Now that you have all the cards in hand, it's up to you!

But where do I start? The server or the client?

It is best that you start with PHP code to make sure that it works well, it will save you a lot of trouble debugging.

In case of malfunctions in your code, be sure to look at the error console and also to check what you got the server, because the error can come from the server and not necessarily from the client.