ΕΠΛ425

Τεχνολογίες Διαδικτύου

(Internet Technologies)

Asynchronous JavaScript And XML (AJAX)

Διδάσκων
Δρ. Χριστόφορος Χριστοφόρου
christophoros@cs.ucy.ac.cy

Goals

- **AJAX** for asynchronous communication between the client and the server.

*Used for **sending** and **receiving** information (i.e., form data) in various formats (**JSON**, XML, HTML, and text files) through **HTTP Messages**.*

Submitting the form

- **Data transfer** from browser to server can be performed via:
  - HTML form submission (the **default way;** we saw this in the previous lecture)
  - **AJAX (Asynchronous JavaScript and XML)**

- **Both methods send** the **form data** to a **PHP file** on the server side but **in a different way**.

Remember….

- Since PHP is running on the server, before continuing you have to **make sure** that **PHP** is **installed** on your PC and the **System Variables Path** is **set correctly**!
- For this, follow these steps to download and configure PHP on you PC: https://www.geeksforgeeks.org/how-to-install-php-in-windows-10/
- Then, go to the folder of your web site and **start the PHP server** using the following command.

```
PHP –S  localhost:8000
```

AJAX - Asynchronous **J**avaScript **A**nd **X**ML

- AJAX is not a programming language → It can be considered as a **client side mechanism** used for **exchanging data between** Front-End and Back-End **over HTTP messages.**

- AJAX is a **misleading name**. It is worth noting that while XML was initially used in AJAX, other data formats such as **JSON (JavaScript Object Notation)** are now **more commonly used**.

AJAX - Asynchronous JavaScript And XML

AJAX works by utilizing **three main components**:

- The browser built-in **XMLHttpRequest object**

  const httpRequest = new XMLHttpRequest();

- This object **allows** JavaScript to:
  - **Make requests** to the web server in the background without disrupting the user's interaction with the page.
  - **Receive data** from the server: Included in ***httpRequest.responseText property***
  - **Send data** to the server for processing: Using ***httpRequest.send()***

AJAX - Asynchronous JavaScript And XML

- **JavaScript**, which **makes requests** and **processes** the **response received** from the server and **updates** the page content in a way that does not require the entire page to be reloaded
  - JavaScript can manipulate the HTML content of the page, modify CSS styles, or perform any other dynamic updates based on the server response.

AJAX - Asynchronous JavaScript And XML

- The **HTML Document Object Model (DOM)**, which represents the structure of the web page as a tree-like structure that can be **manipulated using JavaScript**.

- AJAX **updates** the **page content** by **dynamically changing the DOM tree**, which **enables parts** of the page to **be updated without requiring a full page refresh**

```
document.getElementById("myDiv").innerHTML = "New content";
```

AJAX - Asynchronous JavaScript And XML

- **In a nutshell**, AJAX is the **use** of the **XMLHttpRequest object** **to** **communicate** and **make requests** **with the server**.
- It can **send** (POST) and **receive** (GET) information in **various formats**, including **JSON**, XML, HTML, and text files.
- AJAX's most appealing characteristic is its **"asynchronous" nature!**
- Specifically it **allows web pages** to be **updated asynchronously** by **exchanging data** with a **web server** **behind the scenes**. This means that it is **possible** to **update** parts of a web page, **without reloading** the whole page.

AJAX - Asynchronous JavaScript And XML

- When performing requests from the browser, the two main **HTTP methods** you'll use for data transport are **GET** and **POST**.
- The **main difference** between these two methods lies in the **way data gets sent** to the **web server** application.
  - HTTP **GET** **passes data** to the server application in **name=value pairs**. These **request parameters** are **appended to the URL after ? and separated with &**.
  - HTTP **POST** **passes data** to the server application in the **message body** of the HTTP request, instead of in the URL → With AJAX, the data can be in **various formats** such as plain text, HTML, XML, JSON, or binary data (i.e., in case we upload files).

Remember - HTTP GET Request Message Example

www.chris.com/EPL425/getStudents.php?username=chris&id=3

Query Parameters passing for data filtering

Resource URI

Request line (GET, POST commands)

```
GET /EPL425/getStudents.php?username=chris&id=3 HTTP/1.1
Host: www.chris.com
User-agent: Mozilla/4.0
Connection: close
Accept-language: *
```

Header lines

Entity-Body
Empty when GET method is used

When using HTTP **GET**, the data is sent as a series of key=value pairs, appended to the URL after ?

Remember - HTTP POST Request Message Example

www.chris.com/EPL425/getStudents.php

Resource URI

Request line
(GET, POST commands)

```
POST /EPL425/getStudents.php HTTP/1.1
Host: www.chris.com
User-agent: Mozilla/4.0
Connection: close
Accept-language: *

username=chris&id=3
```

Header lines

Entity-Body
With POST method data are included here

When using HTTP **POST**, the data is sent as a series of **key=value pairs**, similar to HTTP GET. However, instead of appending the parameters to the URL, they are included in the message body.

HTTP POST Request Message Example – with AJAX

www.chris.com/EPL425/getStudents.php

Resource URI

Request line
(GET, POST commands)

```
POST /EPL425/getStudents.php HTTP/1.1
Host: www.chris.com
User-agent: Mozilla/4.0
Connection: close
Accept-language: *

{"username":"chris","id":3}
```

Header lines

Entity-Body
With POST method data are included here

With **AJAX**, we can also send data in JSON format.

Submitting the form: Using Default Way VS using AJAX

- The **default way** of submitting a form causes the **entire page** to be **reloaded**, including **all scripts**, **stylesheets**, and **images**. This approach is known as a **synchronous form submission**.
- On the other hand, **using AJAX** (Asynchronous JavaScript And XML) to submit a form allows you to **send data** to the **server without reloading the entire page**.
- Instead, **only a portion** of the **page** is **updated dynamically**, based on the **response** from the server. This approach is known as an **asynchronous** form submission.

Submitting the form: Using Default Way VS using AJAX

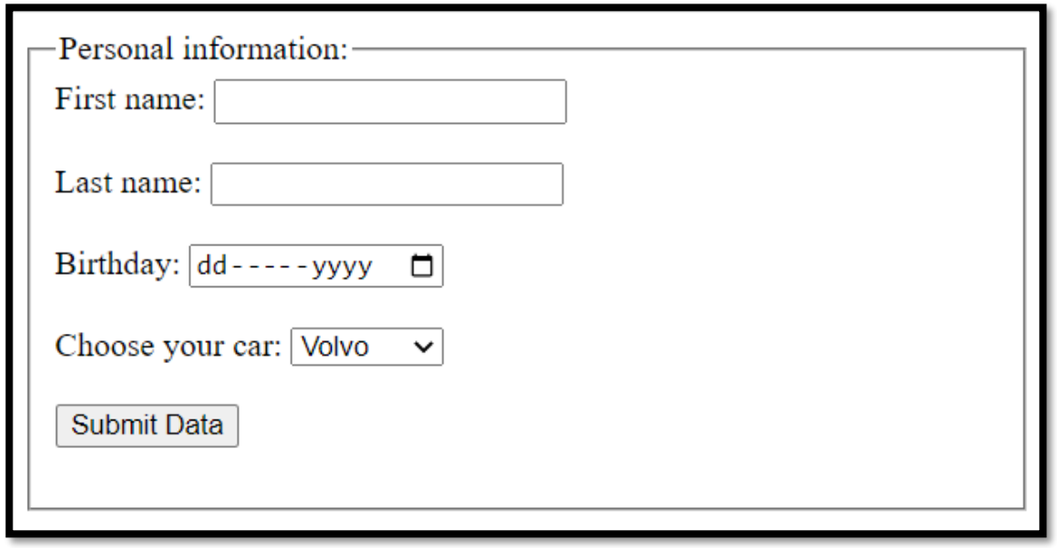
- Using AJAX can provide a **smoother** and **more responsive user experience**, as it allows you to **update parts** of a page **without the need to reload the entire page.**
  - *When you submit a form using the **default synchronous approach**, the browser **sends the form data** to the server and **then waits** for the **server to process the data and send back a response**. This response usually consists of a **new HTML page**, which the **browser then loads and displays**.*
  - *This process can be **slow** and **disruptive** to the **user experience**, especially if the form submission requires a lot of data to be processed or if the server response takes a **long time** to arrive. **During this time**, the user may be **left waiting**, **unsure** whether the **submission has been successful or not**.*
  - *In contrast, **using AJAX to submit a form** allows you to **send data** to the server **in the background**, without interrupting the user's current interaction with the page. This means that the user **can continue** to **interact** with **other parts of the page while the form data** is **being submitted and processed**.*
  - *Once the server has processed the data and sent back a response, the **JavaScript code running** on the page can **update the relevant parts** of the **page dynamically**, **without the need to reload the entire page**. This provides a **smoother** and **more responsive user experience**, as the user **can see the results** of their form submission without **having to wait** for a **new page to load**.*

Submitting the form: Using Default Way VS using AJAX

- Using AJAX can also **reduce the amount of data transferred** between the client and the server, as only the **necessary data** is sent back and forth.
- However, using AJAX **requires additional JavaScript code** to handle the form submission, which can make the **implementation more complex** than a traditional form submission.
- It also requires **a server-side script** that can **handle AJAX requests** and **return data** in a **format** that can be easily processed by JavaScript, such as **JSON** or XML.

Accessing the form's data

- Lets see first how the form's data shown at the right side:
  - Can be **accessed using JavaScript** and **stored** in a **JavaScript object**.
- The **HTML code** for this form as well as the **JavaScript code** accessing the form's data, are shown in the next slides.



```html
<head>
  <title>The first Input Form</title>
  <script src="JS/jsCodeForm.js" defer></script>
</head>

<body>
  <form id="form1">
    <fieldset>
      <legend>Personal information:</legend>
      <label>First name: </label>
      <input type="text" name="firstname" id="firstname" /> <br><br>
      <label>Last name: </label>
      <input type="text" name="lastname" id="lastname" /> <br><br>
      <label for="birthday">Birthday:</label>
      <input type="date" id="birthday" name="birthday" /> <br><br>
      <label for="cars">Choose your car:</label>
      <select id="cars" name="cars">
        <option value="volvo">Volvo</option>
        <option value="peugeot">Peugeot</option>
        <option value="BMW">BMW</option>
        <option value="audi">Audi</option>
      </select> <br><br>

      <button type="button" id="btn1">Submit Data</button> <br><br>
    </fieldset>
  </form>
</body>

</html>
```

When we use AJAX for submitting the form's data we SHOULD USE

<button type="button"> that when clicked will **"fire"** an **"onclick" event** that **when captured will execute the associated JavaScript code.**

If we use

<input type="submit"> or

<button type="submit">

when clicked will automatically **"fire"** a **'submit' event** that **when captured** the **page is refreshed, unless you explicitly prevent it!**

You may prevent this through **event.preventDefault()**.

See the **Additional slides** at the end to see how to do this.

JS/jsCodeForm.js

```
// Here we add a 'click' event listener on the button. After
// the 'Submit Data' button is clicked a 'click' event is fired and
// the submitValues method is triggered.
var btn1 = document.getElementById("btn1");
btn1.addEventListener('click', submitValues);

// submitValues method is the event listener function that receives
// a notification (an object that implements the Event Interface)
// when an event of the specified type (i.e., 'click') occurs.
function submitValues() {
    // Below we create an object to store our data. This object
    // can be later, using JSON.stringify, transformed to JSON format
    const person = {
        fname: document.getElementById("firstname").value,
        lname: document.getElementById("lastname").value,
        bdate: document.getElementById("birthday").value,
        car: document.getElementById("cars").value
    };

    // For debugging purposes we write the data to the Console.
    // Data are written in JSON format. For this JSON.stringify is used.
    console.log(JSON.stringify(person));
}
```

Here we create an object literal to store our form data.

This object can be later, using **JSON.stringify**, transformed to JSON format

Personal information:

First name: Chris

Last name: Christophorou

Birthday: 11-Oct-1978

Choose your car: Peugeot

Submit Data

Console

top

Filter

Default levels

No Issues

```
{"fname":"Chris","lname jsCodeForm.js:25
":"Christophorou","bdate":"1978-10-
11","car":"peugeot"}
```

In JSON format

```javascript
// Here we add a 'click' event listener on the button. After
// the 'Submit Data' button is clicked a 'click' event is fired and
// the submitValues method is triggered.
var btn1 = document.getElementById("btn1");
btn1.addEventListener('click', submitValues);

// submitValues method is the event listener function that receives
// a notification (an object that implements the Event Interface)
// when an event of the specified type (i.e., 'click') occurs.
function submitValues() {
    // The FormData(form) constructor in JavaScript creates a new
    // FormData object from an HTML form element. This object allows
    // you to easily gather form data, including file uploads,
    // and send it to a server using AJAX.
    var form = document.getElementById("form1");
    var formData = new FormData(form);

    const person = {
        fname: formData.get("firstname"),
        lname: formData.get("lastname"),
        bdate: formData.get("birthday"),
        car: formData.get("cars")
    };

    // For debugging purposes we write the data to the Console.
    // Data are written in JSON format. For this JSON.stringify is used.
    console.log(JSON.stringify(person));
}
```

JS/jsCodeForm.js

Another way of collecting the form's data is using **new FormData(form);**
To get the values of form data in JavaScript, you can use the **formData.get()** method, which retrieves the value of a specified field in the form by its **name attribute.**

Note: The **formData** object is a special type of object that is used to create and send HTTP requests that contain **binary data**, such as **files** or **images**, **along** with **other form data**. Thus, you **CANNOT USE JSON.stringify()** on a **FormData** object directly.

Personal information:

First name: Chris

Last name: Christophorou

Birthday: 11-Oct-1978

Choose your car: Peugeot

Submit Data

Console

top

Filter

Default levels

No Issues

```
{"fname":"Chris","lname jsCodeForm.js:25
":"Christophorou","bdate":"1978-10-
11","car":"peugeot"}
```

We have the same result.

Using AJAX – A Step by Step process

The XMLHttpRequest object

- To **make** an **HTTP request to the server** with JavaScript, as a **first step** you **need to create an instance** of an **object** with the **necessary functionality**.
- This is where **XMLHttpRequest** object comes in.

```
const httpRequest = new XMLHttpRequest();
```

Using AJAX – A Step by Step process

The XMLHttpRequest object

- The XMLHttpRequest (XHR) object provides a **range** of **methods** and **properties** that you can use to **create**, **send**, and **handle HTTP requests** in JavaScript.

- Some of the **most commonly used methods** and **properties** of the XHR object include:
  - **.open(…): Opens** a **new HTTP request** with the **specified method** (e.g. GET, POST), **URL**, and *optional* async flag, username, and password.
  - **.send(…): Sends the HTTP request** to the server, *optionally* including **data** in the request body.

Using AJAX – A Step by Step process

The XMLHttpRequest object

- **.onreadystatechange**: A **property associated** with an **event handler (i.e., a function)** that is **called** whenever the **.readyState property** (see next slide) **changes** during the HTTP request.

- **.responseText**: A property that contains the server's response data as a string.

Using AJAX – A Step by Step process

The XMLHttpRequest object

- **.readyState:** A property that indicates at the Client the **current state** of the **HTTP request** and can have the following values:

Value	State	Description
0	UNSENT	The XMLHttpRequest client has been created, but the open() method hasn't been called yet.
1	OPENED	open() method has been invoked. During this state, the request headers can be set using the setRequestHeader() method and the send() method can be called which will initiate the fetch.
2	HEADERS_RECEIVED	send() has been called, and headers and status are available.
3	LOADING	Downloading; **responseText** holds partial data.
4	DONE	The fetch operation is complete. This could mean that either the data transfer has been completed successfully or failed.

The XMLHttpRequest.readyState property returns the state an **XMLHttpRequest client** is in.

Using AJAX – A Step by Step process

How to make an HTTP request ("POST")

- **.status**: A **property** that indicates the **HTTP status code** of the **server response**. Some of the most commonly used HTTP status codes are:

Status	Description
200 OK	The request was successful
201 Created	The request was successful, and a new resource was created.
204 No Content	The request was successful, but there is no content to return
301 Moved Permanently	The requested resource has been permanently moved to a new URL.
302 Found	The requested resource has been temporarily moved to a new URL.
400 Bad Request	The server could not understand the request due to invalid syntax.
401 Unauthorized	The request requires authentication.
403 Forbidden	The server understood the request, but is refusing to fulfill it.
404 Not Found	The requested resource could not be found.
500 Internal Server Error	A generic error message, indicating that something went wrong on the server while processing the request.
503 Service Unavailable	The server is currently unavailable due to maintenance or overload.

Using AJAX – A Step by Step process

- Next, **you need** to **actually "create" the HTTP request**, by **calling** the **open() method** of the **httpRequest object** you created**.**

```
httpRequest.open('POST', 'PHP/get_data.php', true);
```

The **first parameter** is the **HTTP request method** like "GET", "POST", "PUT", "DELETE". Keep the method all-capitals, otherwise some browsers (like Firefox) might not process the request.

The **second parameter** is a string **representing the URL (relative or absolute)** to **send the request to (i.e., in this example the "get_data.php"** file located in the **PHP folder** of our web site).

The **third parameter** (boolean) is **optional** and sets whether the **request is asynchronous.** If **true** (this is the default), **JavaScript execution** will **continue** and the **user can interact with the page** while the **server response has yet to arrive** (notification of a completed transaction is provided using **event listeners)**.

***Setting** **async** to **false** means that **the statement you are calling has to complete before the next statement in your function, can be called**. If you set async: true then that statement will begin it's execution and the next statement will be called regardless of whether the async statement has completed yet.*

Using AJAX – A Step by Step process

The **syntax** of **open()**:

```
open(method, url)
open(method, url, async)
open(method, url, async, user)
open(method, url, async, user, password)
```

The **user name** and **password** are **optional parameters** that can be used for **authentication purposes**. By default these values are null.

Note: Always use asynchronous requests!!! **Synchronous requests** on the main thread can be easily **disruptive** to the **user experience** and **should be avoided**; in fact, **many browsers** have **deprecated synchronous XHR support** on the main thread entirely. Synchronous requests are permitted in Web Workers.

Using AJAX – A Step by Step process

- After a request is made to the server, you will **receive** a **response back**. Thus, before you send the request, you **need to tell** the **XMLHttpRequest object** you created, **which JavaScript function will handle** the **response from the server.**

- This is done by **setting** the **onreadystatechange property** of the **httpRequest** object that we created, to the **function** that **will be invoked** when the **request changes state.**

```
httpRequest.onreadystatechange = handleResponse;
```

```
function handleResponse() {
    // Process the server response here.
}
```

Note: There are **NO parentheses** or **parameters after the function name**, because you're **assigning a reference to the function, rather than actually calling it**.

Using AJAX – A Step by Step process

- Next, **you need** to **actually send the form data** by **calling** the **send() method** of the **`httpRequest` object.**
- Form's data **should be sent** in a **format** that the **server can parse**. In this case we send the data in **JSON format**.

```
const person = {
        fname: document.getElementById("firstname").value,
        lname: document.getElementById("lastname").value,
        bdate: document.getElementById("birthday").value,
        car: document.getElementById("cars").value
     };

data = JSON.stringify(person);

httpRequest.send(data);
```

Using AJAX – A Step by Step process

Note: If you want to "POST" data, you may have to set the MIME type of the request. For example, if you are sending data in **JSON format** you can use the following, before calling **send()**.

Note that this is **not mandatory** if the php code is yours, since you already now how to parse the data on the server.

```
httpRequest.setRequestHeader('Content-Type', 'application/json');
```

```
$content_type = $_SERVER['CONTENT_TYPE'];
if ($content_type === 'application/json') {
    // The request contains JSON data
    // Get the JSON raw data using file_get_contents("php://input");
    // Parse the JSON data using json_decode() function
}
```

This is how you check in the php script the content type of the data sent

Using AJAX – A Step by Step process

- **Remember**: When we created the **httpRequest object**, we provided the name of a JavaScript function to **handle the response**.

```
httpRequest.onreadystatechange = handleResponse;
```

```
function handleResponse() {
    // Process the server response here.
}
```

What should this function do?

Using AJAX – A Step by Step process

First, the function needs to check the **state** of the request and the HTTP response status codes of the HTTP response.
If the **readystate** has the value of **XMLHttpRequest.DONE,** and the **status** has the value of **200**, that means that the fetch operation is **complete** and the **data transfer** has been **completed successfully**.

```
function handleResponse() {

    if (httpRequest.readyState === XMLHttpRequest.DONE && httpRequest.status === 200) {
            // Assuming server sends a JSON format we are using JSON.parse() to
            // convert the JSON string back to an object
              var myObject = JSON.parse(httpRequest.responseText);

            // Then do what ever you want with the data in myObject
    }
}
```

Using AJAX – A Step by Step process

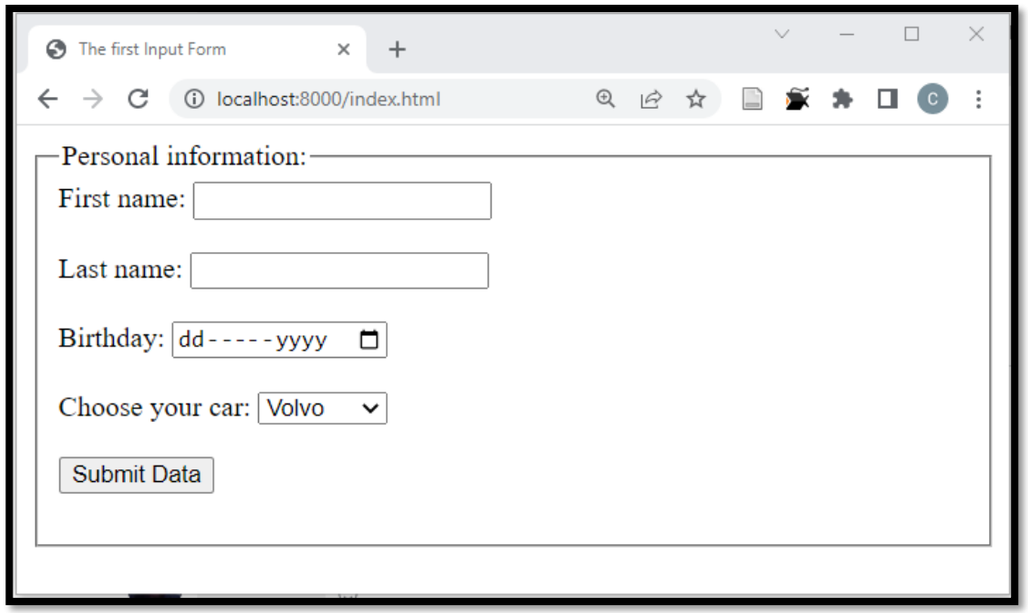
Then you **can access the data the server sent!!!** This **httpRequest.responseText property** includes a **string** that contains the **response** to the **request as text;** or **null** if the request was unsuccessful or has not yet been sent. Specifically **httpRequest.responseText** includes what was **echo** by the PHP script and in this case we assume that is in **JSON format.**

```
function handleResponse() {

    if (httpRequest.readyState === XMLHttpRequest.DONE && httpRequest.status === 200) {
            // Assuming server sends a JSON format we are using JSON.parse() to
            // convert the JSON string back to an object
              var myObject = JSON.parse(httpRequest.responseText);

            // Then do what ever you want with the data in myObject
    }
}
```

Using AJAX – A Step by Step process

Putting these all together (making a "POST" request)

- In this example we make a "**POST**" request to the web server using AJAX, to send some personal information about a user.

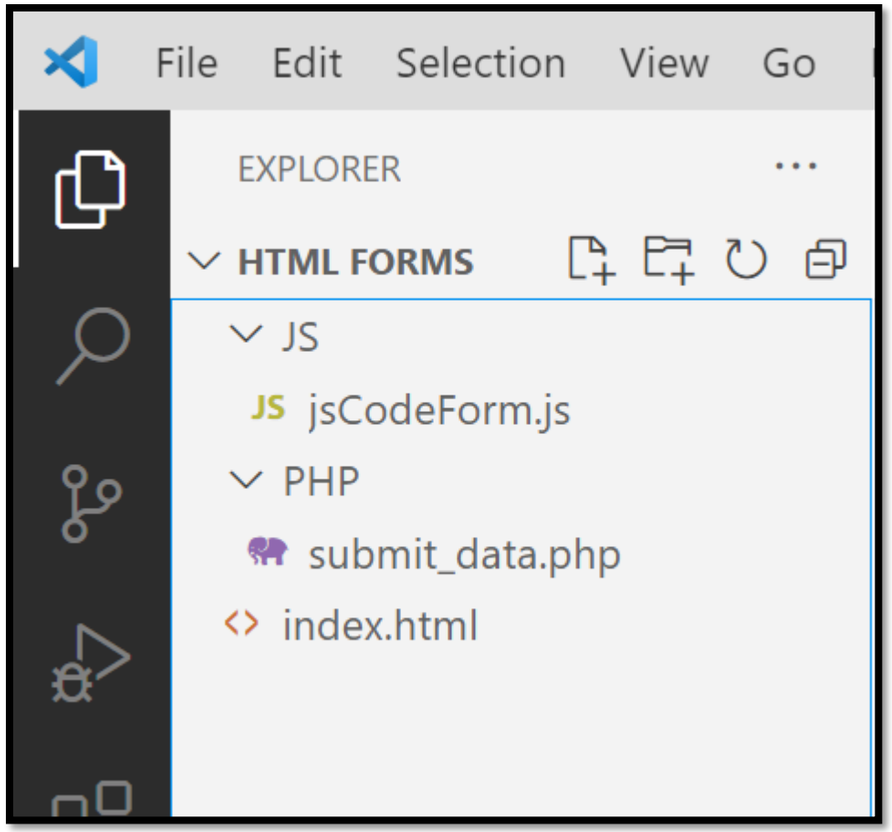


Note that what is shown in the next slides is the classic way to do AJAX, but it's not only way. Two other ways to make a POST request is using **Fetch API** and **jQuery**.

Using AJAX – A Step by Step process

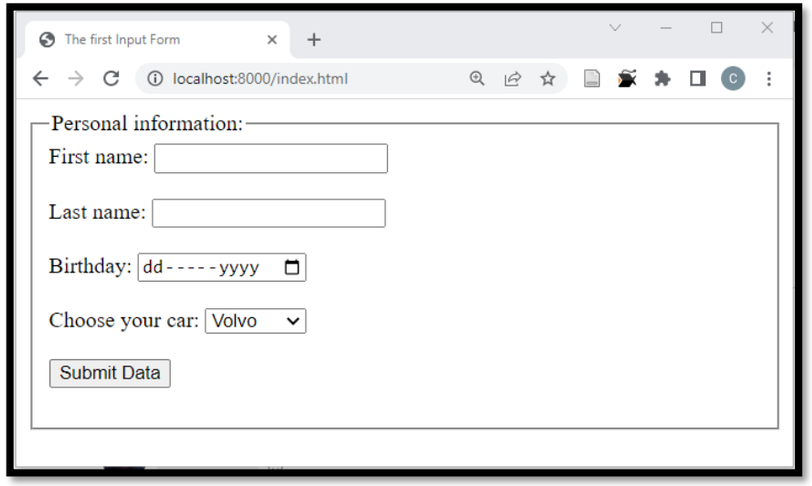
Putting these all together (making a "POST" request)

- The structure of our web site is shown below:



index.html

```html
<!DOCTYPE html>
<html>
<head>
    <title>The first Input Form</title>
    <script src="JS/jsCodeForm.js" defer></script>
</head>

<body>
    <form id="form1">
        <fieldset>
            <legend>Personal information:</legend>
            <label>First name: </label>
            <input type="text" name="firstname" id="firstname" /> <br><br>
            <label>Last name: </label>
            <input type="text" name="lastname" id="lastname" /> <br><br>

            <label for="birthday">Birthday:</label>
            <input type="date" id="birthday" name="birthday" /> <br><br>

            <label for="cars">Choose your car:</label>
            <select id="cars" name="cars">
                <option value="volvo">Volvo</option>
                <option value="peugeot">Peugeot</option>
                <option value="BMW">BMW</option>
                <option value="audi">Audi</option>
            </select> <br><br>

            <button type="button" id="btn1">Submit Data</button> <br><br>
        </fieldset>
    </form>
    <!-- We will use this p HTML Element to print the status to the user after submission -->
    <p id="status"></p>
</body>
</html>
```



jsCodeForm.js

```
// Here we add a 'click' event listener on the button. After the 'Submit Data' button
// is clicked a 'click' event is fired and the postValues method is triggered.
var btn1 = document.getElementById("btn1");
btn1.addEventListener('click', postValues);

// We declare the httpPostRequest here so as to be visible in all our code
var httpPostRequest;

// In this example we will use AJAX to "POST" the values using JSON to the server
function postValues() {

    const person = {
        fname: document.getElementById("firstname").value,
        lname: document.getElementById("lastname").value,
        bdate: document.getElementById("birthday").value,
        car: document.getElementById("cars").value
    };

    const data = JSON.stringify(person);

    httpPostRequest = new XMLHttpRequest();
    httpPostRequest.open('POST', 'PHP/submit_data.php', true);
    httpPostRequest.onreadystatechange = handleResponse;

    httpPostRequest.setRequestHeader('Content-Type', 'application/json');
    httpPostRequest.send(data);
}
```

Here we create an object to store our data. This object is then, using JSON.stringify(), transformed to JSON format string.

Here we use XHR to post the data to the web server to a php file called **submit_data.php** stored in the PHP folder of our web site

We also set the request header, telling the server how to process the data we send over. We will tell to the server that this will be a JSON payload.

Then we sent the body (i.e., the **data**) of our "POST" request.

If **everything went well**, access the response data included in **httpPostRequest.responseText** and save it to a variable. This message will then be displayed to the user in the browser.

Note: If you are expecting JSON use **JSON.parse(responseMessage)** to convert it back to an object and then you can access and do whatever you want with the values.

```
function handleResponse() {

    if (httpPostRequest.readyState === XMLHttpRequest.DONE && httpPostRequest.status === 200) {
        var responseMessage = httpPostRequest.responseText;
        document.getElementById("status").innerHTML = responseMessage;
    }
    else {
        // There was a problem with the request.
        // For example, the response may have a 404 (Not Found)
        // or 500 (Internal Server Error) response code.
    }
}
```

jsCodeForm.js (Continue)

THE HTTP POST Request Message will send the data in JSON format to the PHP file

http://localhost:8000/

Resource URI

Request line
(GET, POST commands)

```
POST /'PHP/submit_data.php  HTTP/1.1
Host: localhost:8000
User-agent: Mozilla/4.0
Connection: close
Accept-language: *

{"fname":"Christophoros","lname
":"Christophorou","bdate":"1978
-10-11","car":"peugeot"}
```

Header lines

Entity-Body
With POST method data are included here

In this example we send data in JSON format.

```php
<?php
// Here we will received the HTTP POST request, read the body that
// includes the data and then convert it to a php object

// First we create a new variable and get the payload from the request
// using file_get_contents("php://input")
$requestPayload = file_get_contents("php://input");

// Then we convert the JSON payload to php object (using json_decode)
$personObject = json_decode($requestPayload);

// From now on you can take the person data submitted and save it in the Database using MySQL.

// Now send the response back (echo) to the form that submitted the Request.
// This message will be stored in the httpRequest.responseText
// The status 200 OK is also send back before that.
echo "The person " .  $personObject->fname . " is successfully stored in the Database!";
?>
```

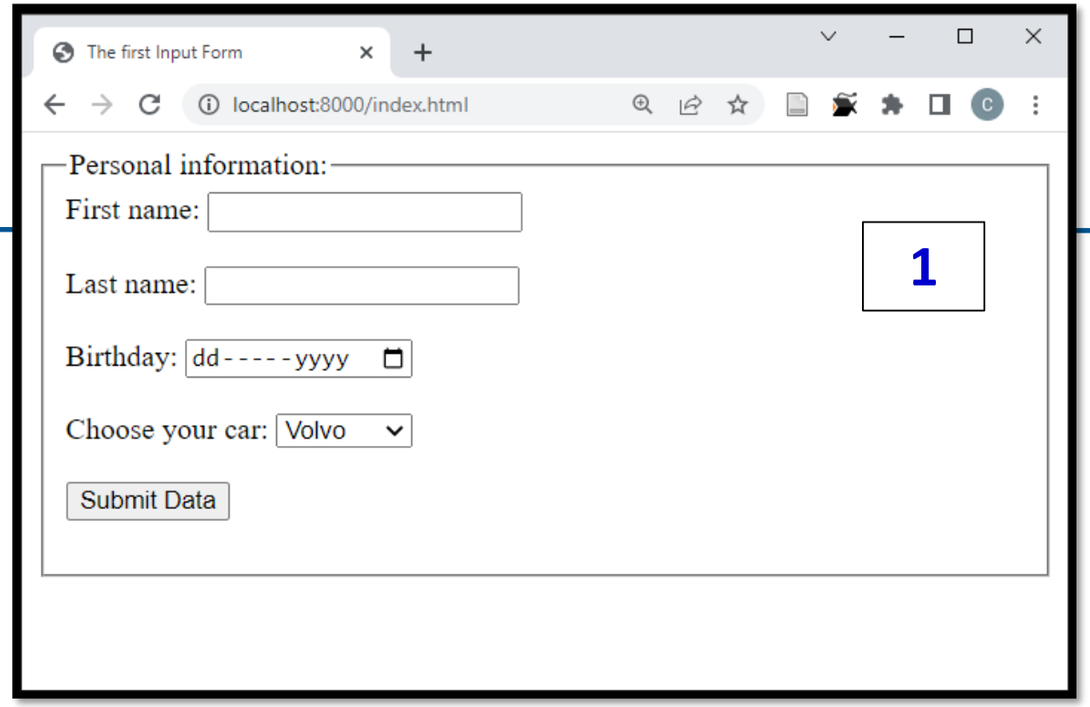
submit_data.php

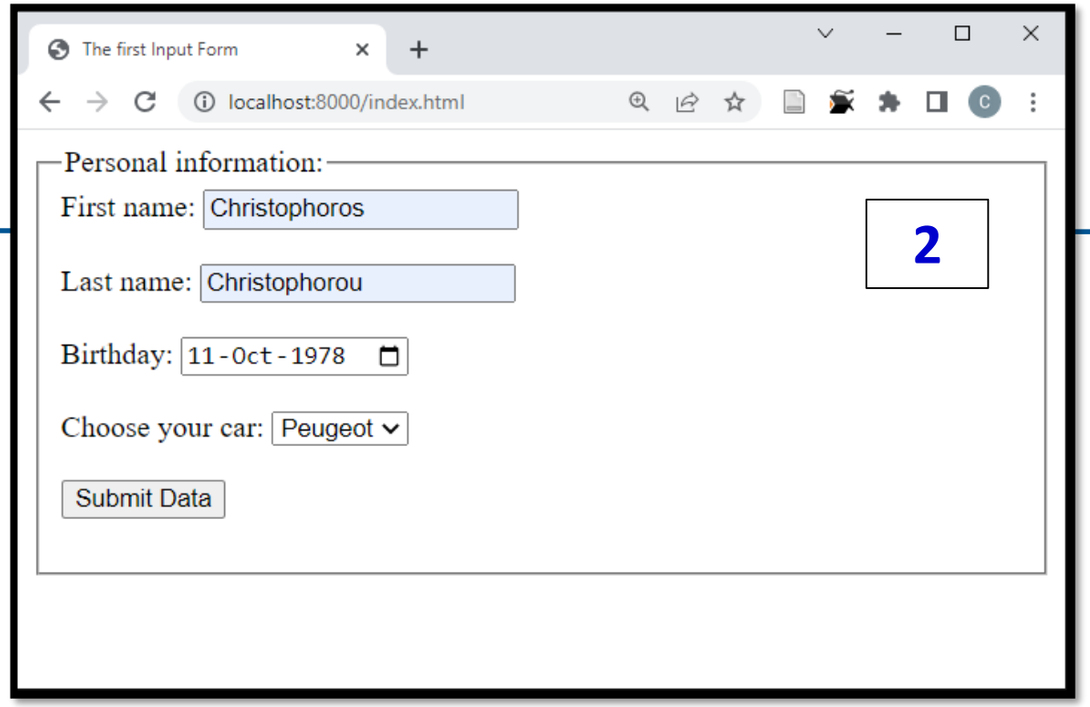
Note: To access the **JSON data** sent via a POST request using AJAX in PHP, you can use the **file_get_contents()** function to read the **raw HTTP request body** and then parse the JSON data using the **json_decode()** function.

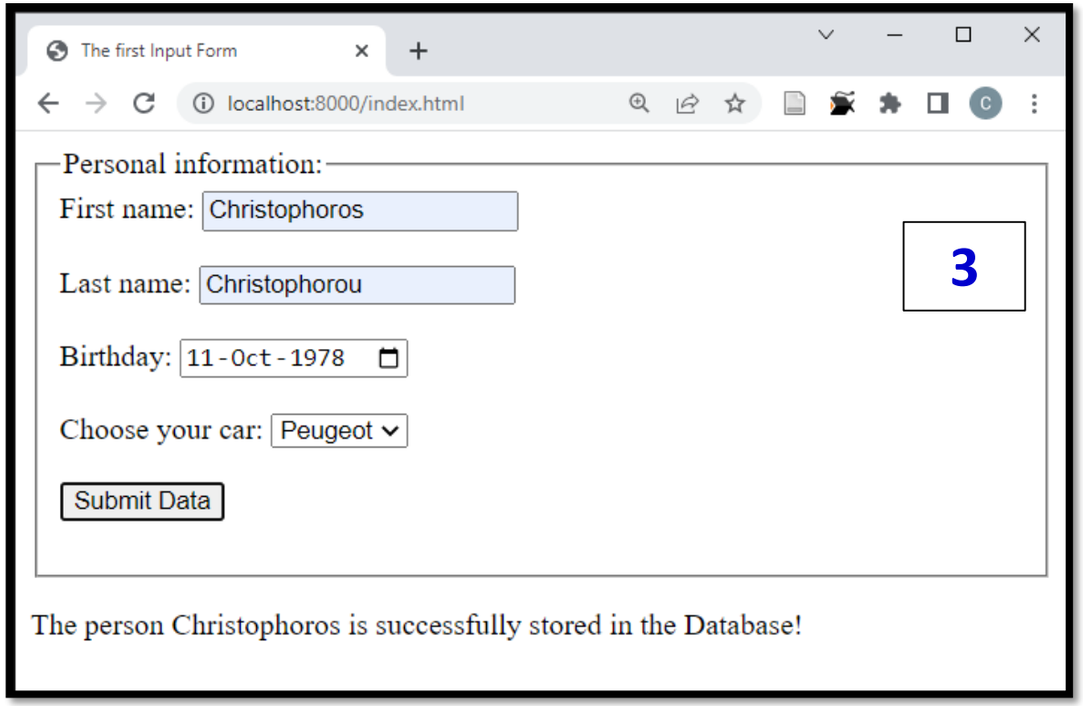
Note: To declare a variable in php you start with $. To access a php object property we use **->** . Also to join two strings in php we use the dot . The following is also correct but you need to use double quotes:

```php
echo "The person $personObject->fname is successfully stored in the Database!";
```

We will see more about PhP and MySQL in later lectures!


The first Input Form
localhost:8000/index.html
Personal information:
First name:
Last name:
Birthday: dd-----yyyy
Choose your car: Volvo
Submit Data
1

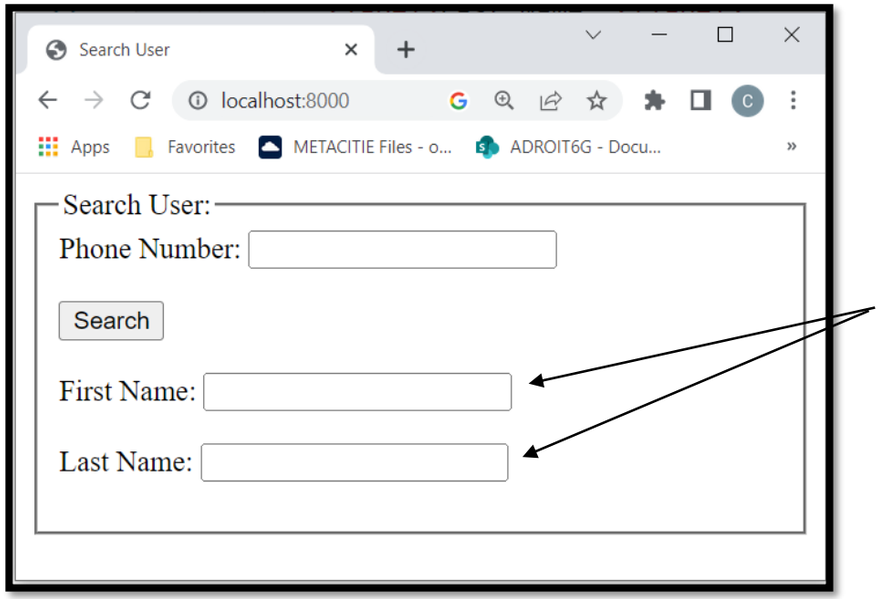

The first Input Form
localhost:8000/index.html
Personal information:
First name: Christophoros
Last name: Christophorou
Birthday: 11-Oct-1978
Choose your car: Peugeot
Submit Data
2


The first Input Form
localhost:8000/index.html
Personal information:
First name: Christophoros
Last name: Christophorou
Birthday: 11-Oct-1978
Choose your car: Peugeot
Submit Data
3
The person Christophoros is successfully stored in the Database!

Using AJAX – A Step by Step process

Putting these all together (making a "GET" request)

- In this example we make a "**GET**" request to the web server using AJAX, to search for a person details, using as a query data the person's phone number.

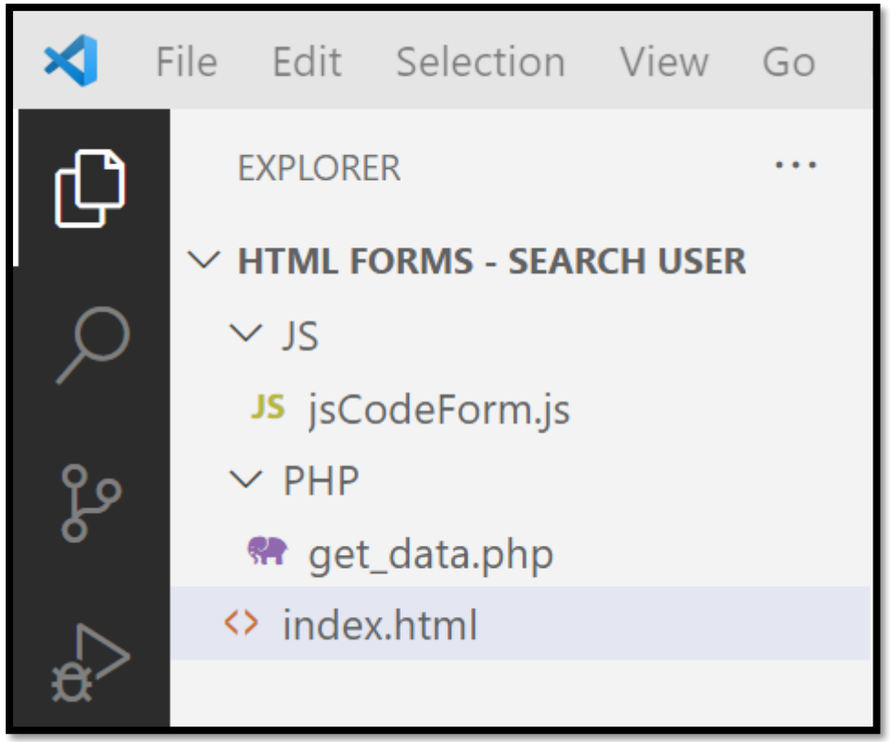


The person details we search for will be included here

Using AJAX – A Step by Step process

Putting these all together (making a "GET" request)

- The structure of our web site is show below:



index.html

```html
<!DOCTYPE html>
<html>

<head>
    <title>Search User</title>
    <script src="JS/jsCodeForm.js" defer></script>
</head>

<body>
    <form id="form1">
        <fieldset>
            <legend>Search User:</legend>
            <label>Phone Number: </label>
            <input type="text" name="pNumber" id="pNumber" /> <br><br>

            <button type="button" id="btn1">Search</button> <br><br>

            <!-- We will use the following input form elements to store the person details received -->
            <label>First Name: </label>
            <input type="text" name="fName" id="fName" readonly/> <br><br>
            <label>Last Name: </label>
            <input type="text" name="lName" id="lName" readonly/> <br><br>
        </fieldset>
    </form>
</body>

</html>
```

jsCodeForm.js

```
var btn = document.getElementById("btn1");
btn.addEventListener('click', getValues);

var httpGetRequest;

function getValues() {

    // Below we create the query string that will be added to the url after ?
    var qData = "pNumber=" + document.getElementById("pNumber").value;

    // Now we will use XHR to sent the query data using "GET" to the web server to a
    // php file called get_data.php. Since we are sending information using GET
    // we add this information to the URL after ?.
    httpGetRequest = new XMLHttpRequest();
    httpGetRequest.open('GET', 'PHP/get_data.php?' + qData, true);
    httpGetRequest.onreadystatechange = handleResponse;

    // Then we sent our request. Note that "GET" messages do not have a body,
    // thus we cannot include any data as an input parameter in .send()
    httpGetRequest.send();
}
```

```
// Here we define what will happen when the response from the Server is received
function handleResponse() {
    if (httpGetRequest.readyState === XMLHttpRequest.DONE && httpGetRequest.status === 200) {

        // Access the response data and check if a person is actually found
        if (httpGetRequest.responseText === "Not Found!") {
            document.getElementById("fName").value = "Not Found!";
            document.getElementById("lName").value = "Not Found!";
        }
        else {
            // Access the response data, parse it to a JS object and save it to a variable.
            var responseMessage = JSON.parse(httpGetRequest.responseText);

            // Display the data to the browser
            document.getElementById("fName").value = responseMessage.fName;
            document.getElementById("lName").value = responseMessage.lName;
        }
    }
    else {
        // There was a problem with the request.
        // For example, the response may have a 404 (Not Found)
        // or 500 (Internal Server Error) response code.
    }
}
```

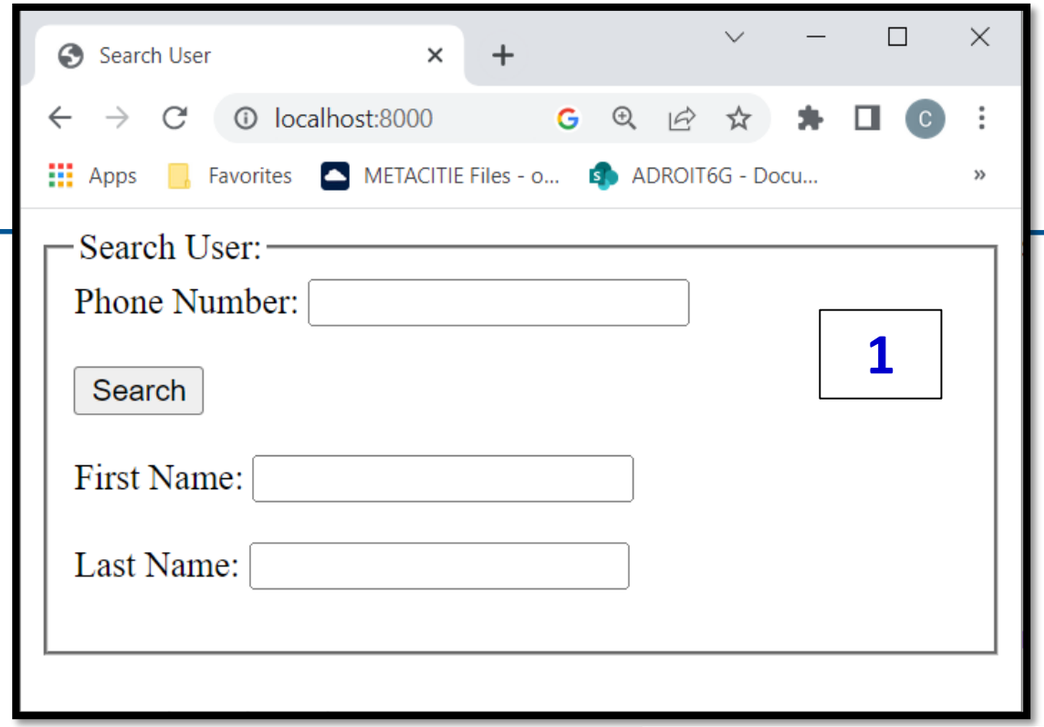
get_data.php

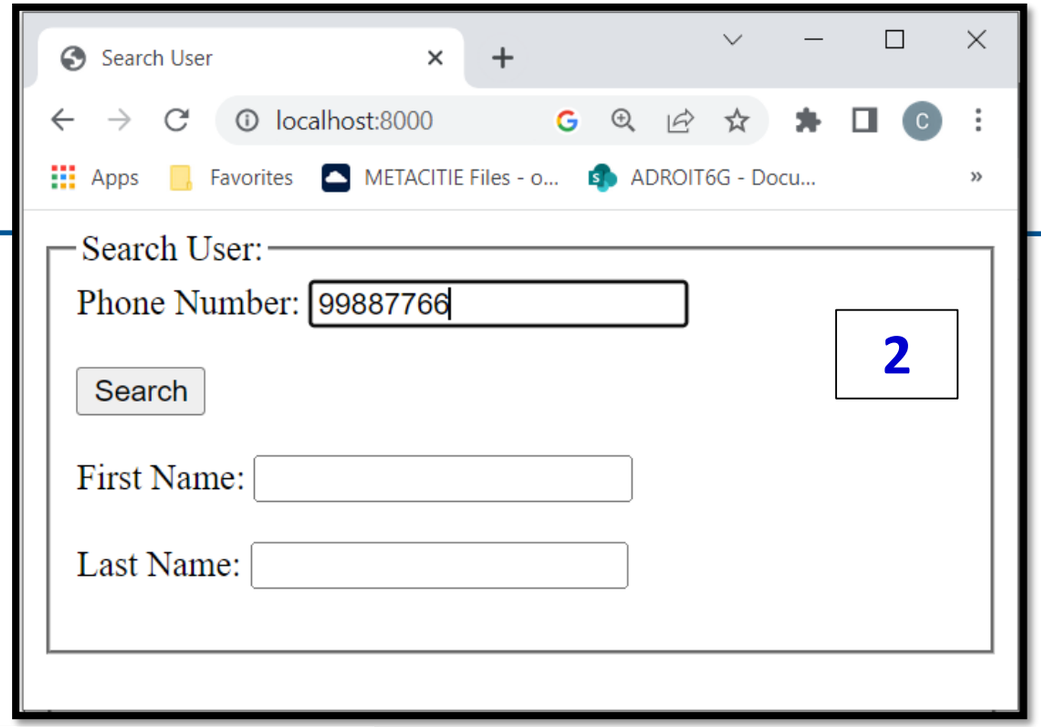
```php
<?php

class Person
{
    // Properties
    public $fName;
    public $lName;
    public $pNumber;

    //constructor function
    function __construct($fName, $lName, $pNumber)
    {
        $this->fName = $fName;
        $this->lName = $lName;
        $this->pNumber = $pNumber;
    }
}

// Initialize an array with persons. We will assume that this
// array of persons is our database.
$persons = array();
$persons[0] = new Person("Christophoros", "Christophorou", "99887766");
$persons[1] = new Person("Andonis", "Christophorou", "99554433");
$persons[2] = new Person("Markos", "Christophorou", "99667788");
$persons[3] = new Person("Chanma", "Christophorou", "99334455");
```

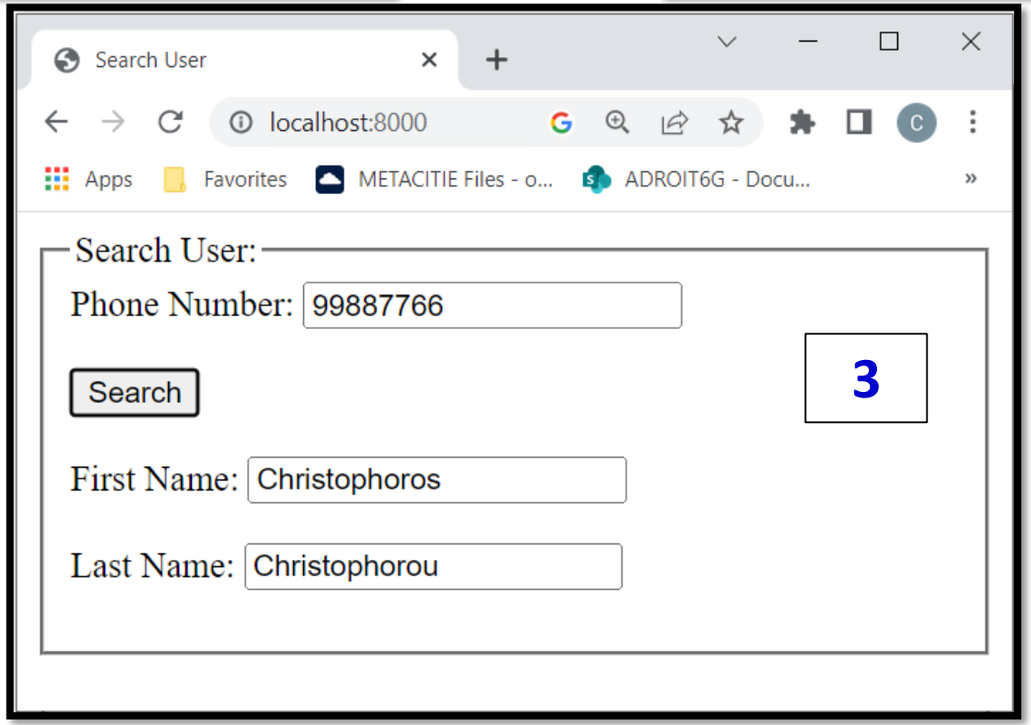
Here we create the class Person, that we will use to create some person objects that we will include in an array and use as a database

```php
// Here we will receive the HTTP GET request and catch the query string
// (in this case is pNumber). Then check for each person included in
// the "database" which of these users has this pNumber and echo the
// person object in JSON format (using json_encode($person)). If a person
// is not found echo "Not Found!"
$pNumber = $_GET["pNumber"];
$found = False;
if ($pNumber !== '') {
    foreach ($persons as $person) {
        if (strcmp($person->pNumber, $pNumber) === 0) {
            // sent the details in JSON format
            echo json_encode($person);
            $found = True;
            break;
        }
    }
}

if ($found === False)
    echo "Not Found!";
?>
```

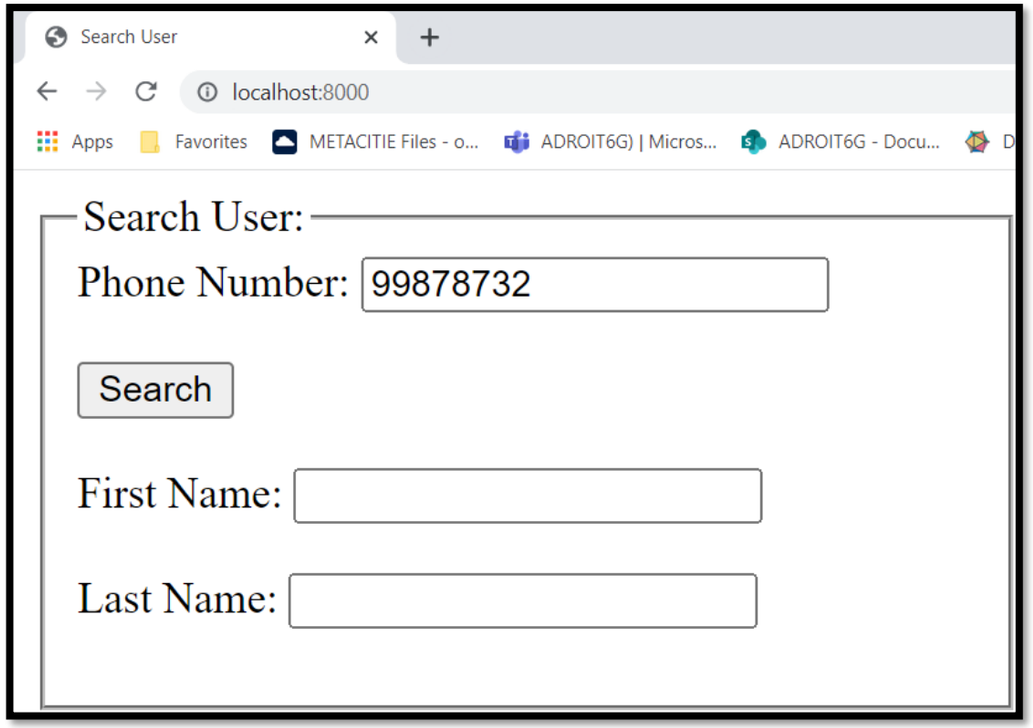
get_data.php (Continued)

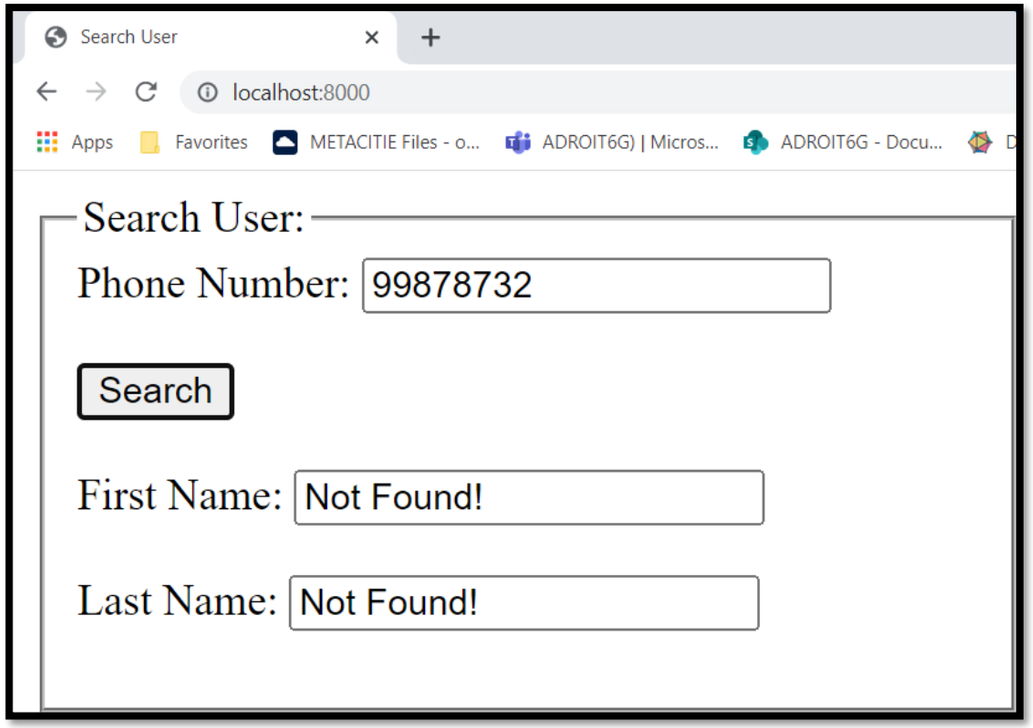






This will be the Result!


Search User
localhost:8000
Apps
Favorites
METACITIE Files - o...
ADROIT6G) | Micros...
ADROIT6G - Docu...
Search User:
Phone Number:
99878732
Search
First Name:
Last Name:


Search User
localhost:8000
Apps
Favorites
METACITIE Files - o...
ADROIT6G) | Micros...
ADROIT6G - Docu...
Search User:
Phone Number:
99878732
Search
First Name:
Not Found!
Last Name:
Not Found!

Using AJAX – A Step by Step process

Putting these all together (making a "GET" request)

- An **alternative way** is to **add** in the URL string the **query data** in **JSON format** and then **process the query** in the **get_data.php file accordingly**!

- *The index.html file we keep the same.*

- *We modify slightly the code in the **jsCodeForm.js** and the **get_data.php** files.*

jsCodeForm.js

```
var btn = document.getElementById("btn1");
btn.addEventListener('click', getValues);

var httpGetRequest;

function getValues() {

    // Below we create an object to store our query data (in this case is only the phone number).
    const queryData = {
        pNumber: document.getElementById("pNumber").value,
    };

    // Then this object is transformed into JSON (using JSON.stringify())
    // and will be added to the url (after ?) that we will send to the server.
    qData = JSON.stringify(queryData);

    // Now we will use XHR to sent the data using "GET" to the web server to a php
    // file called get_data.php. Since we are sending information using GET
    // we add this information to the URL after ?. However, after the ? we
    // also add q= . This q will be used in php (i.e., $_GET["q"]) to
    // catch the JSON format query string.
    httpGetRequest = new XMLHttpRequest();
    httpGetRequest.open('GET', 'PHP/get_data.php?q=' + qData, true);
    httpGetRequest.onreadystatechange = handleResponse;

    // Then we sent our request. Note that "GET" messages do not have a body,
    // thus we cannot include any data as an input parameter in .send()
    httpGetRequest.send();
}
```

We changed this part of the code

```
// Here we define what will happen when the response from the Server is received
function handleResponse() {
    if (httpGetRequest.readyState === XMLHttpRequest.DONE && httpGetRequest.status === 200) {

        // Access the response data and check if a person is actually found
        if (httpGetRequest.responseText === "Not Found!") {
            document.getElementById("fName").value = "Not Found!";
            document.getElementById("lName").value = "Not Found!";
        }
        else {
            // Access the response data, parse it to a JS object and save it to a variable.
            var responseMessage = JSON.parse(httpGetRequest.responseText);

            // Display the data to the browser
            document.getElementById("fName").value = responseMessage.fName;
            document.getElementById("lName").value = responseMessage.lName;
        }
    }
    else {
        // There was a problem with the request.
        // For example, the response may have a 404 (Not Found)
        // or 500 (Internal Server Error) response code.
    }
}
```

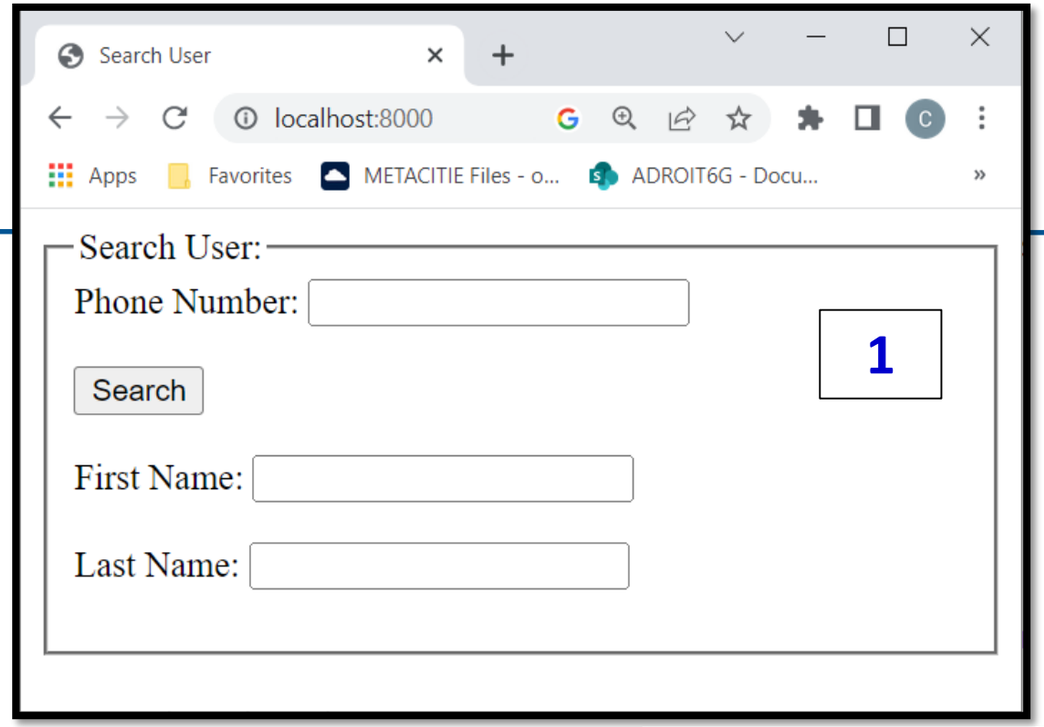
jsCodeForm.js (Continue)

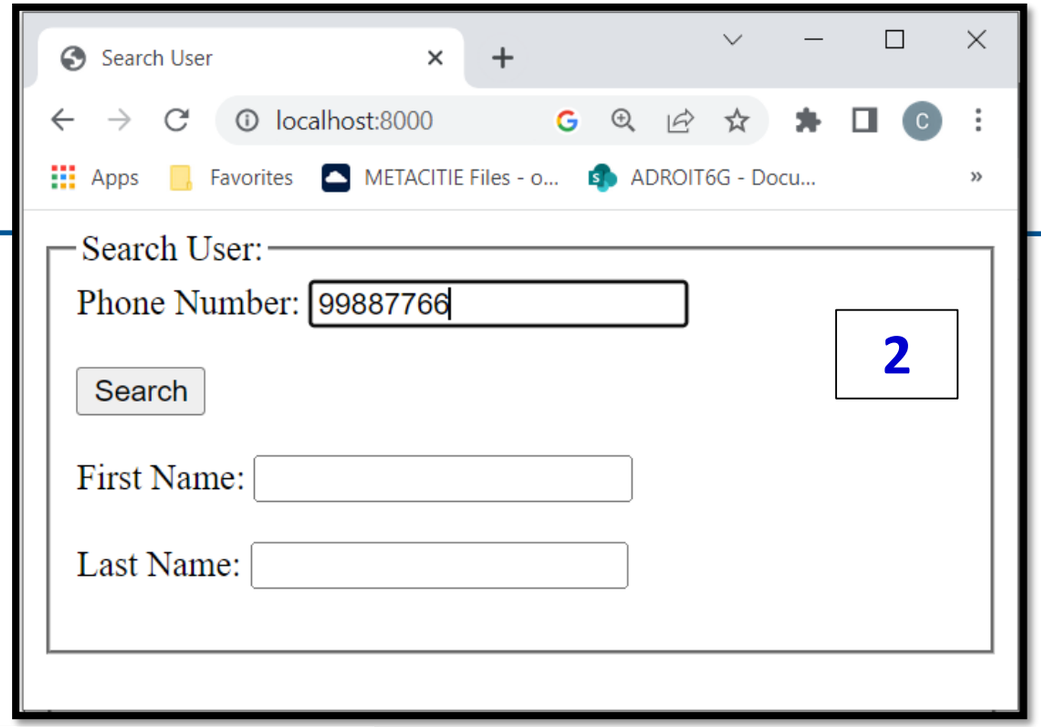
get_data.php

```php
<?php

class Person
{
    // Properties
    public $fName;
    public $lName;
    public $pNumber;

    //constructor function
    function __construct($fName, $lName, $pNumber)
    {
        $this->fName = $fName;
        $this->lName = $lName;
        $this->pNumber = $pNumber;
    }
}

// Initialize an array with persons. We will assume that this
// array of persons is our database.
$persons = array();
$persons[0] = new Person("Christophoros", "Christophorou", "99887766");
$persons[1] = new Person("Andonis", "Christophorou", "99554433");
$persons[2] = new Person("Markos", "Christophorou", "99667788");
$persons[3] = new Person("Chanma", "Christophorou", "99334455");
```

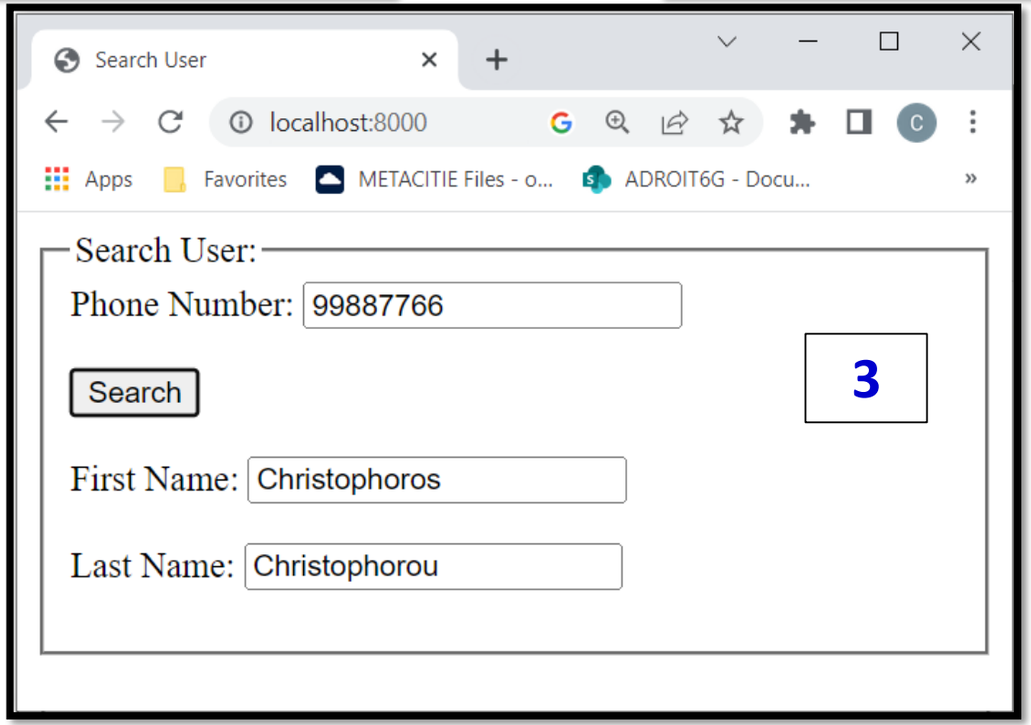
```php
// Here we will receive the HTTP GET request and catch the query string
// (in this case is q that is associated to the JSON text).
// Then, we use json_decode() to parse the JSON text to a PHP object.
// Then check for each person included in the "database" which of these users has
// the  pNumber and echo the person object in JSON format (using json_encode($person)).
// If a person is not found echo "Not Found!"

$q = json_decode($_GET["q"]);
$found = False;
if ($q->pNumber !== '') {
    foreach ($persons as $person) {
        if (strcmp($person->pNumber, $q->pNumber) === 0) {
            // sent the details in JSON format
            echo json_encode($person);
            $found = True;
            break;
        }
    }
}

if ($found === False)
    echo "Not Found!";
?>
```

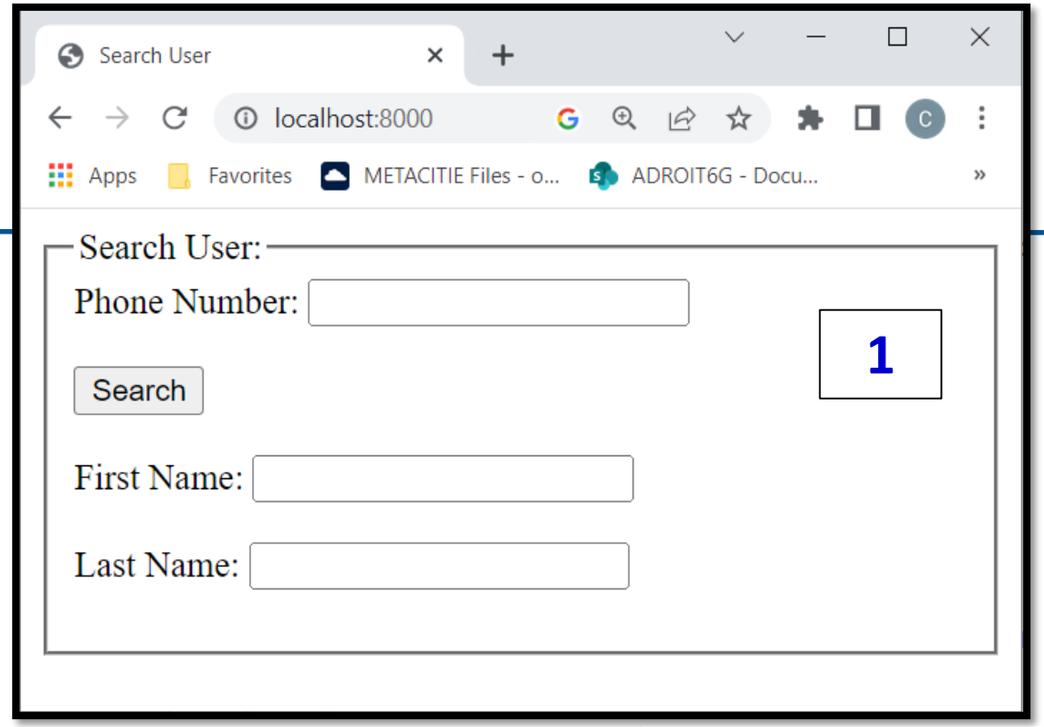
get_data.php (Continued)

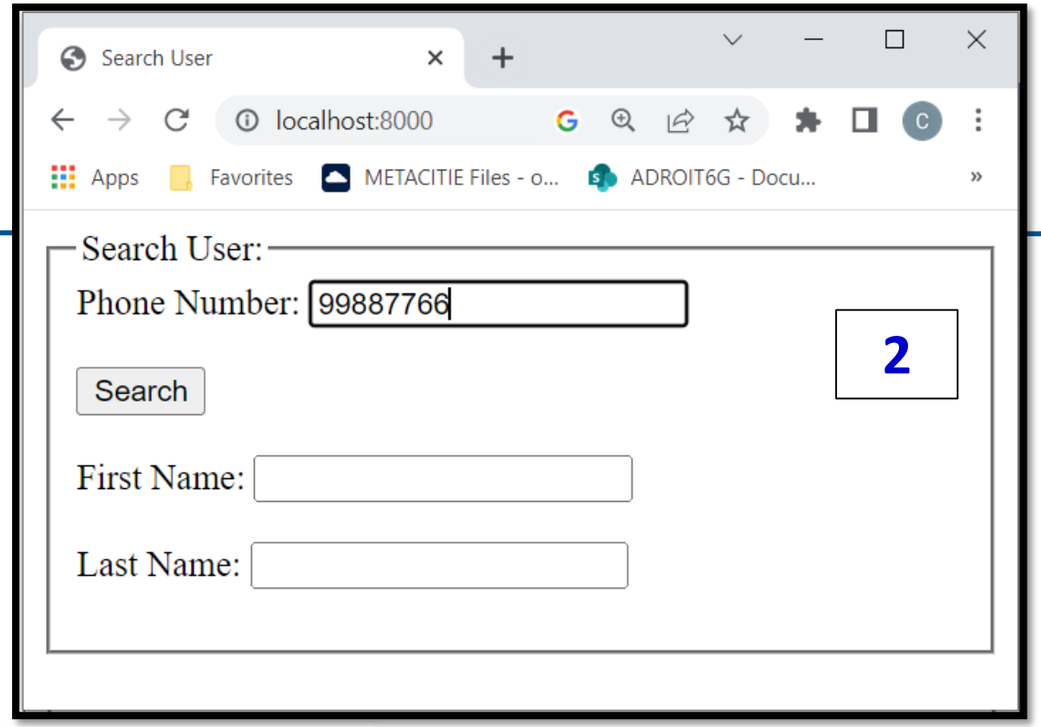




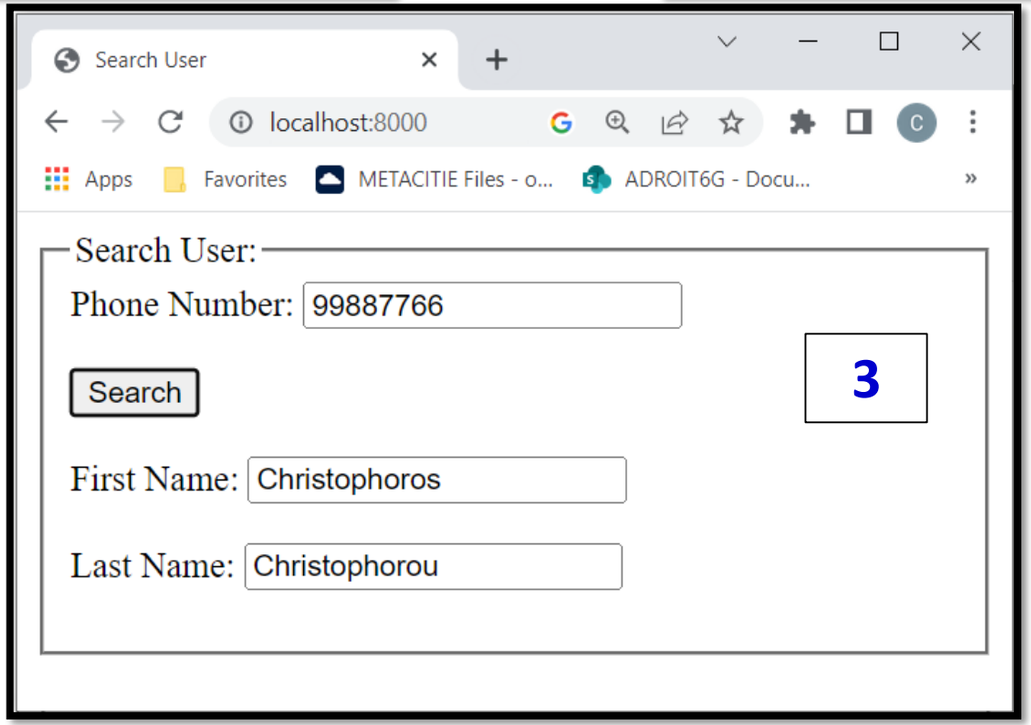
The Result will be the same!







The Result will be the same!

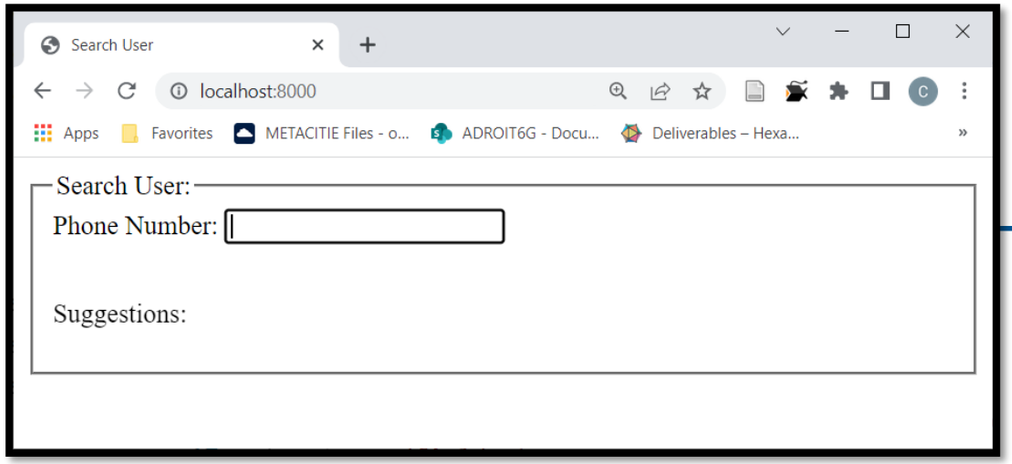


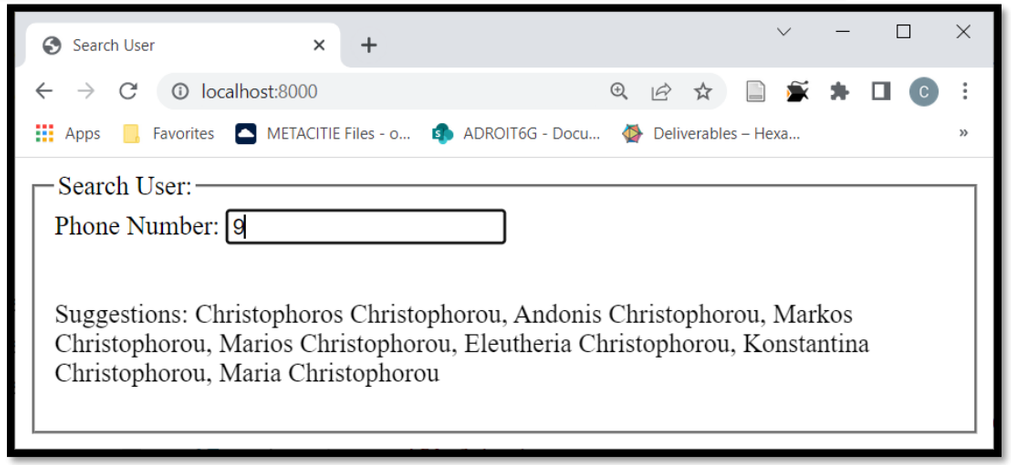
Using AJAX – A Step by Step process

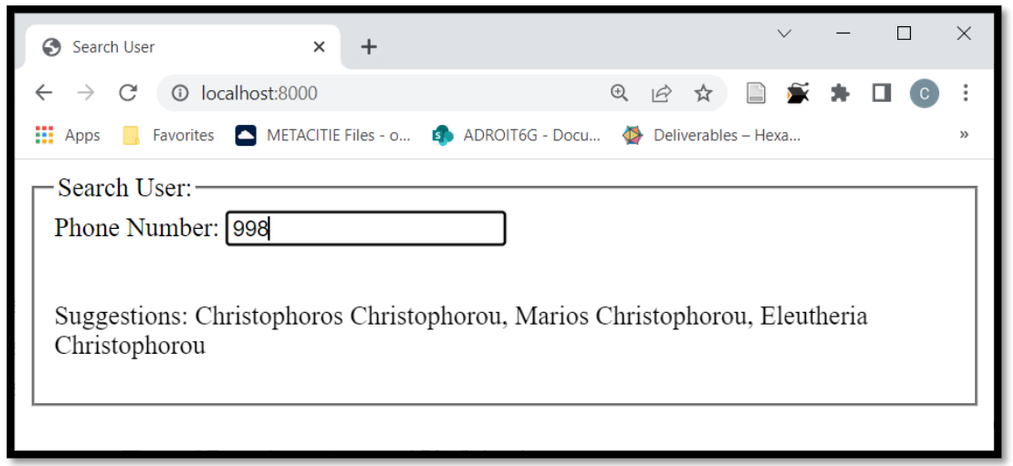
Putting these all together (making a "GET" request)

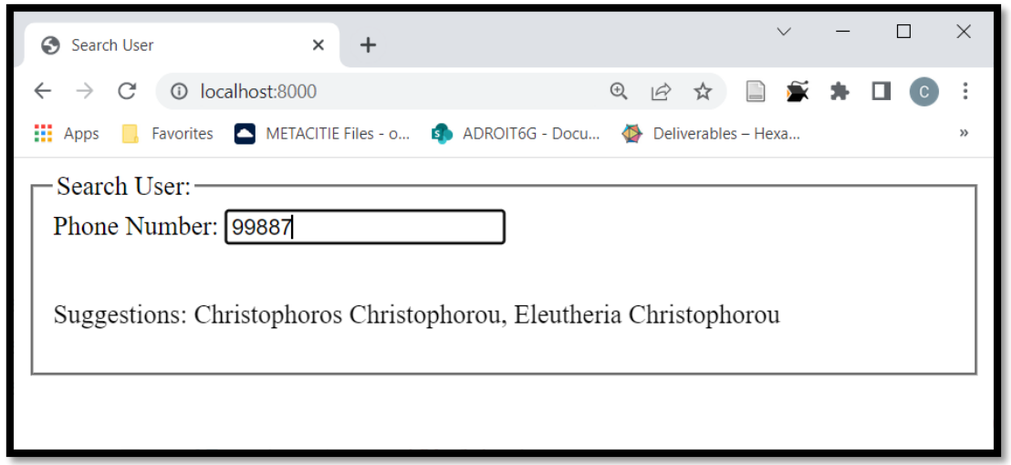
- In the following slides we show an example of how request are done **asynchronously** using an "**onkeyup**" event.
- That is while the user is typing his/her search keywords, suggestions matching the typed keywords, are displayed automatically to the browser.

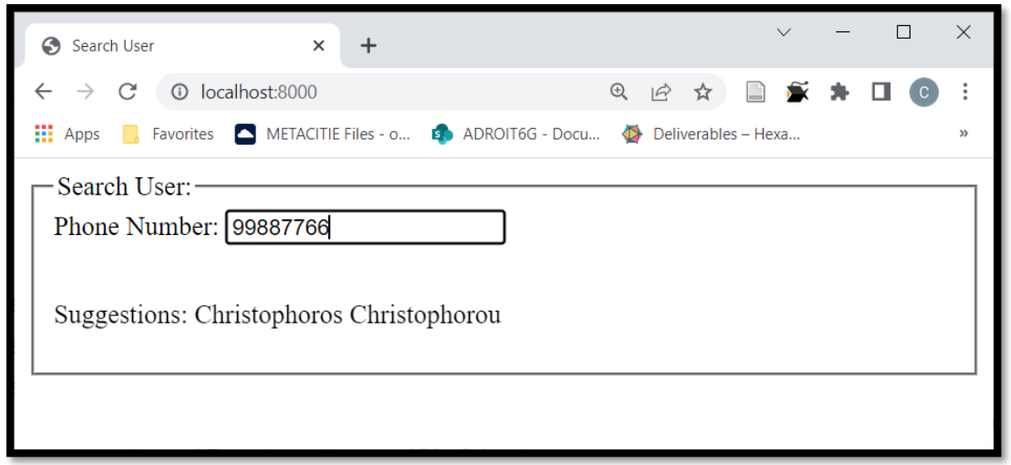
Execution Example







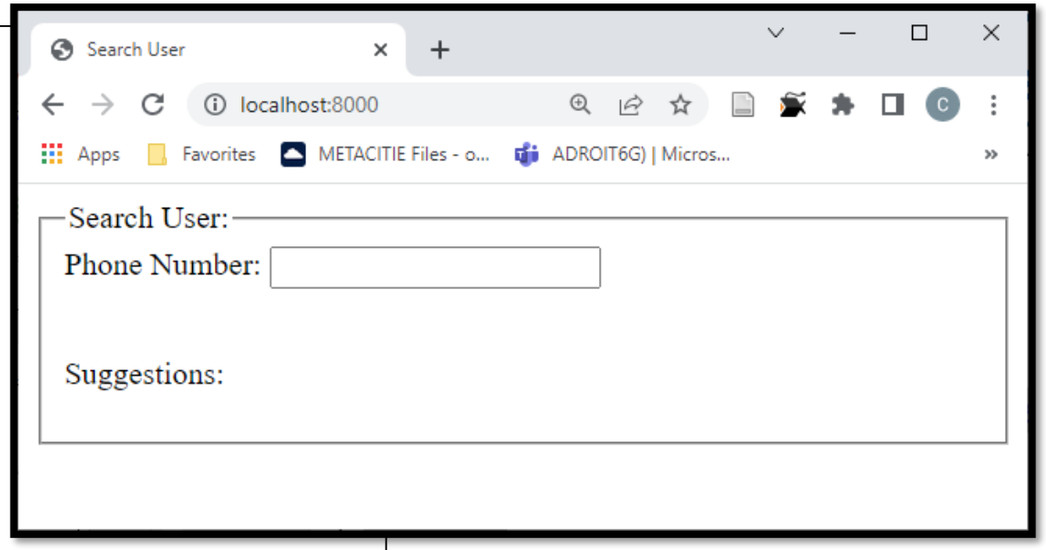




Using AJAX – A Step by Step process

Putting these all together (making a "GET" request for every "keyup" event)

```
<!DOCTYPE html>
<html>

<head>
    <title>Search User</title>
    <script src="JS/jsCodeForm.js" defer></script>
</head>

<body>
    <form id="form1">
        <fieldset>
            <legend>Search User:</legend>
            <label>Phone Number: </label>
            <input type="text" name="pNumber" id="pNumber" /> <br><br>

            <p>Suggestions: <span id="output"></span></p>
        </fieldset>
    </form>
</body>

</html>
```



This is how the form will look in the browser

jsCodeForm.js

```
var searchInput = document.getElementById("pNumber");
searchInput.addEventListener("keyup", getValues);

var httpGetRequest;

function getValues() {

    if (searchInput.value.length == 0){
        document.getElementById("output").innerHTML = '';
    }
    else{
        // Make an AJAX request. For this we will use XHR to sent the data using "GET"
        // to the web server to a php file called get_data.php. Since we are sending
        // information using GET we add this information to the URL after ?. However,
        // after the ? we  also add q= . This q will be used in php (i.e., $_GET["q"]) to
        // catch the search string.

        httpGetRequest = new XMLHttpRequest();
        httpGetRequest.open('GET', 'PHP/get_data.php?q=' + searchInput.value, true);
        httpGetRequest.onreadystatechange = handleResponse;

        // Then we sent our request. Note that "GET" messages do not have a body,
        // thus we cannot include any data as an input parameter in .send()
        httpGetRequest.send();
    }
}
```

```
// Here we define what will happen when the reponse from the Server is received
function handleResponse() {

    if (httpGetRequest.readyState === XMLHttpRequest.DONE && httpGetRequest.status === 200) {
        // Perfect! Request Successfully Received (OK)
        // Access the response data, and display the text included in the
        // span element with id = "output"

        document.getElementById("output").innerHTML = httpGetRequest.responseText;
    }
    else {
        // There was a problem with the request.
        // For example, the response may have a 404 (Not Found)
        // or 500 (Internal Server Error) response code.
    }
}
```

jsCodeForm.js (Continue)

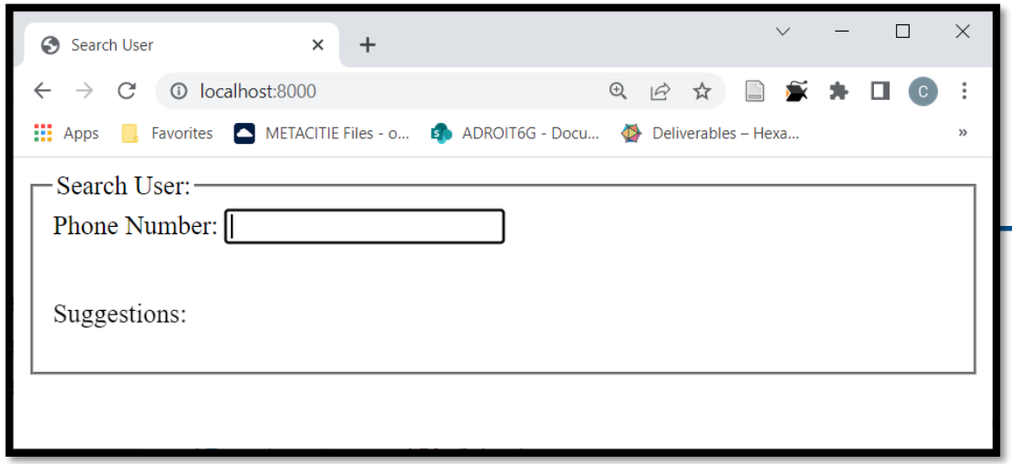
get_data.php

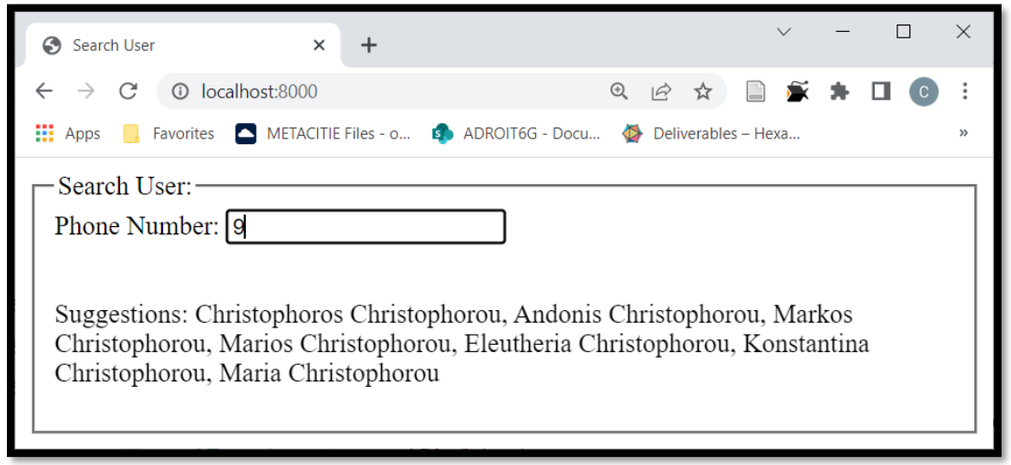
```php
<?php

class Person
{
    // Properties
    public $fName;
    public $lName;
    public $pNumber;

    //constructor function
    function __construct($fName, $lName, $pNumber)
    {
        $this->fName = $fName;
        $this->lName = $lName;
        $this->pNumber = $pNumber;
    }
}

// Initialize an array with persons. We will assume that this
// array of persons is our database.
$persons = array();
$persons[0] = new Person("Christophoros", "Christophorou", "99887766");
$persons[1] = new Person("Andonis", "Christophorou", "99554433");
$persons[2] = new Person("Markos", "Christophorou", "99667788");
$persons[3] = new Person("Chanma", "Christophorou", "99334455");
```

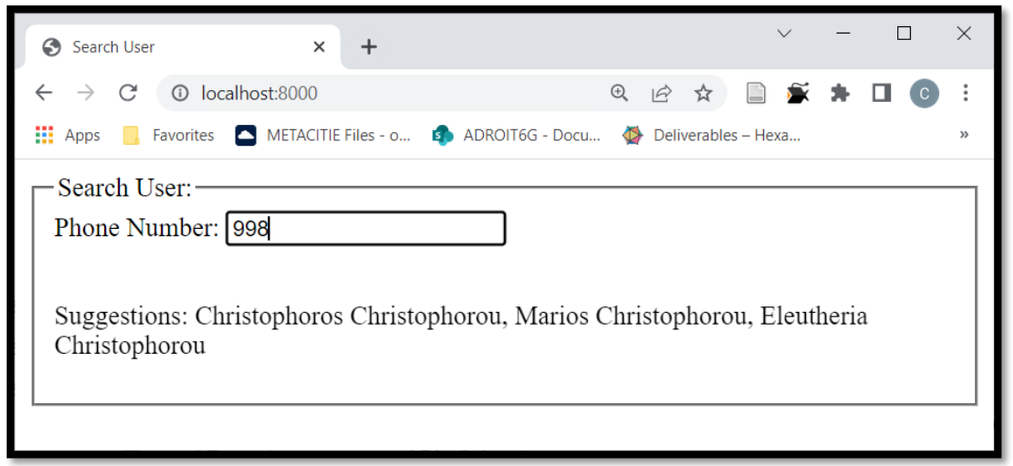
get_data.php (Continue)

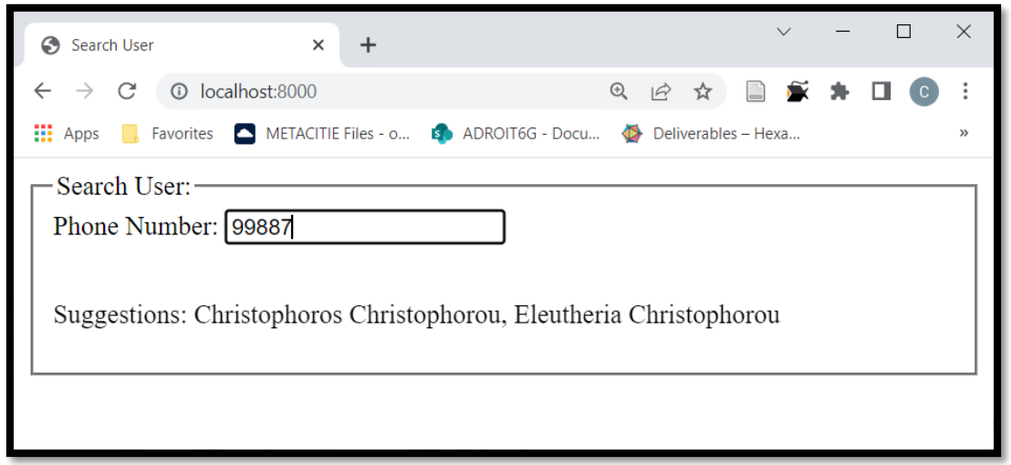
```php
// Here as a first step we will received the HTTP GET request and
// catch the query string (in this case is q that is associated to search input)
$q = $_GET["q"];

// Then we construct the $suggestions variable to include all the persons that the search keyword
// is part of the phone number of the person included in the array persons
$suggestions = '';

if ($q !== '') {
    foreach ($persons as $person) {
        if (strpos($person->pNumber, $q) !== false) {
            if($suggestions === ""){
                $suggestions = $person->fName . " " . $person->lName;
            }
            else{
                $suggestions = $suggestions . ", " . $person->fName . " " . $person->lName;
            }
        }
    }
}

if ($suggestions === ""){
    echo "No Suggestions";
}
else {
    echo ($suggestions);
}

?>
```

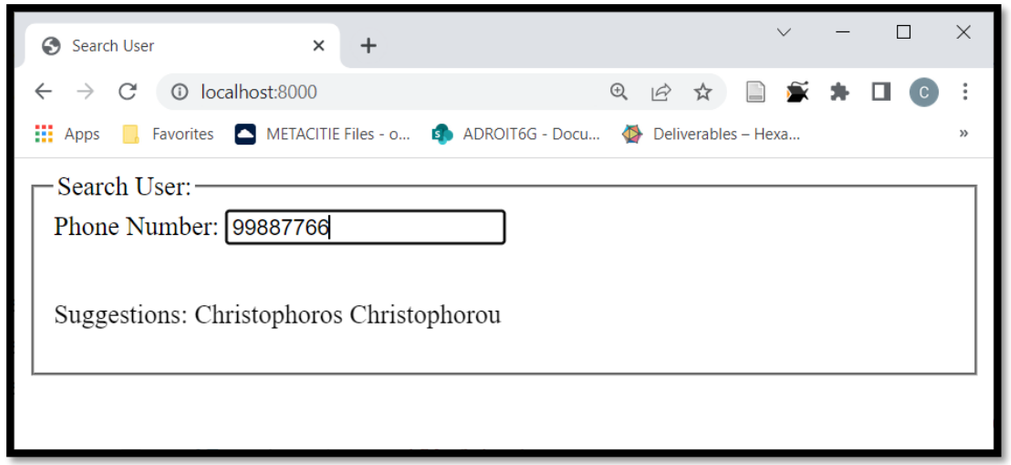
The **strpos(string, substring)** function in PHP is used to **find the position of the first occurrence of a substring in a string**. The function takes two arguments: the string to search within, and the substring to search for.
It **returns** the **position of the first occurrence** of the substring in the string, or **false** if the substring is **not found**.


Search User
localhost:8000
Apps
Favorites
METACITIE Files - o...
ADROIT6G - Docu...
Deliverables – Hexa...
Search User:
Phone Number:
Suggestions:


Search User
localhost:8000
Apps
Favorites
METACITIE Files - o...
ADROIT6G - Docu...
Deliverables – Hexa...
Search User:
Phone Number: 9
Suggestions: Christophoros Christophorou, Andonis Christophorou, Markos Christophorou, Marios Christophorou, Eleutheria Christophorou, Konstantina Christophorou, Maria Christophorou


Search User
localhost:8000
Apps
Favorites
METACITIE Files - o...
ADROIT6G - Docu...
Deliverables – Hexa...
Search User:
Phone Number: 998
Suggestions: Christophoros Christophorou, Marios Christophorou, Eleutheria Christophorou


Search User
localhost:8000
Apps
Favorites
METACITIE Files - o...
ADROIT6G - Docu...
Deliverables – Hexa...
Search User:
Phone Number: 99887
Suggestions: Christophoros Christophorou, Eleutheria Christophorou


Search User
localhost:8000
Apps
Favorites
METACITIE Files - o...
ADROIT6G - Docu...
Deliverables – Hexa...
Search User:
Phone Number: 99887766
Suggestions: Christophoros Christophorou

Ερωτήσεις?

Additional Slides

```html
<!DOCTYPE html>
<html>
<head>
    <title>The first Input Form</title>
    <script src="JS/jsCodeForm.js" defer></script>
</head>

<body>
    <form id="form1">
        <fieldset>
            <legend>Personal information:</legend>
            <label>First name: </label>
            <input type="text" name="firstname" id="firstname" /> <br><br>
            <label>Last name: </label>
            <input type="text" name="lastname" id="lastname" /> <br><br>

            <label for="birthday">Birthday:</label>
            <input type="date" id="birthday" name="birthday" /> <br><br>

            <label for="cars">Choose your car:</label>
            <select id="cars" name="cars">
                <option value="volvo">Volvo</option>
                <option value="peugeot">Peugeot</option>
                <option value="BMW">BMW</option>
                <option value="audi">Audi</option>
            </select> <br><br>

            <input type="submit" value="Submit Data" /> <br><br>
        </fieldset>
    </form>
</body>
</html>
```

Note: If you want to add a **'submit' event listener on the form** (see the JavaScript code in the next slide), **you need to use**

<input type="submit"> or

<button type="submit">.

These, **when clicked** will automatically **"fire"** a **'submit' event** that **when captured will execute the associated JavaScript code.**

You can also use

<button type="button">

but clicking the button **DOES NOT 'fire'** a **'submit' event**.

```javascript
// Here we add a 'submit' event listener on the form. After
// the 'Submit Data' is clicked a 'submit' event is fired and
// the getValues method is triggered.
var form = document.getElementById("form1");
form.addEventListener('submit', getValues);

// getValues method is the event listener function that receives
// a notification (an object that implements the Event Interface)
// when an event of the specified type (i.e., 'submit') occurs.
function getValues(event) {
    // The following command prevents the page from refreshing
    // and loosing the form data.
    event.preventDefault();

    // Below we create an object to store our data. This object
    // can be later, using JSON.stringify, transformed to JSON format
     const person = {
        fname: document.getElementById("firstname").value,
        lname: document.getElementById("lastname").value,
        bdate: document.getElementById("birthday").value,
        car: document.getElementById("cars").value
     };

    // For debugging purposes we write the data to the Console.
    // Data are written in JSON format. For this JSON.stringify is used.
    console.log(JSON.stringify(person));
}
```

Note1: HTML is a **stateless protocol.** This means that it **cannot store anything** and data on the form **will be lost on page refresh**.

Note2: **By default, on form submission**, the **page is refreshed, unless you explicitly prevent it!** You may prevent this through **event.preventDefault()**.

Personal information:

First name: Chris

Last name: Christophorou

Birthday: 11-Oct-1978

Choose your car: Peugeot

Submit Data

Console

top

Filter

Default levels

No Issues

```
{"fname":"Chris","lname jsCodeForm.js:25
":"Christophorou","bdate":"1978-10-
11","car":"peugeot"}
```

In JSON format

```javascript
// Here we add a 'submit' event listener on the form. After
// the 'Submit Data' is clicked a 'submit' event is fired and
// the getValues method is triggered.
var form = document.getElementById("form1");
form.addEventListener('submit', getValues);

// getValues method is the event listener function that receives
// a notification (an object that implements the Event Interface)
// when an event of the specified type (i.e., 'submit') occurs.
function getValues(event) {
    // The following command prevents the page from refreshing
    // and loosing the form data.
    event.preventDefault();

    // The FormData(form) constructor in JavaScript creates a new
    // FormData object from an HTML form element. This object allows
    // you to easily gather form data, including file uploads,
    // and send it to a server using AJAX.

    var formData = new FormData(form);

    // For debugging purposes we write the data to the Console.
    console.log(formData.get("firstname"));
    console.log(formData.get("lastname"));
    console.log(formData.get("birthday"));
    console.log(formD
    console.log(formD
}
```

Another more easy way of collecting the form's data is using **new FormData(form);** To get the values of form data in JavaScript, you can use the **formData.get()** method, which retrieves the value of a specified field in the form by its **name attribute.**

Note: The **FormData** object is a special type of object that is used to create and send HTTP requests that contain **binary data**, such as **files** or **images**, **along** with **other form data**. Thus, you **CANNOT USE JSON.stringify()** on a **FormData** object directly.

Personal information:

First name: Chris

Last name: Christophorou

Birthday: 11-Oct-1978

Choose your car: Peugeot

Submit Data

Console

top

Filter

Default levels

No Issues

Chris	jsCodeForm.js:21
Christophorou	jsCodeForm.js:22
1978-10-11	jsCodeForm.js:23
peugeot	jsCodeForm.js:24

Each value separately

Accessing the form's data - using event.preventDefault()

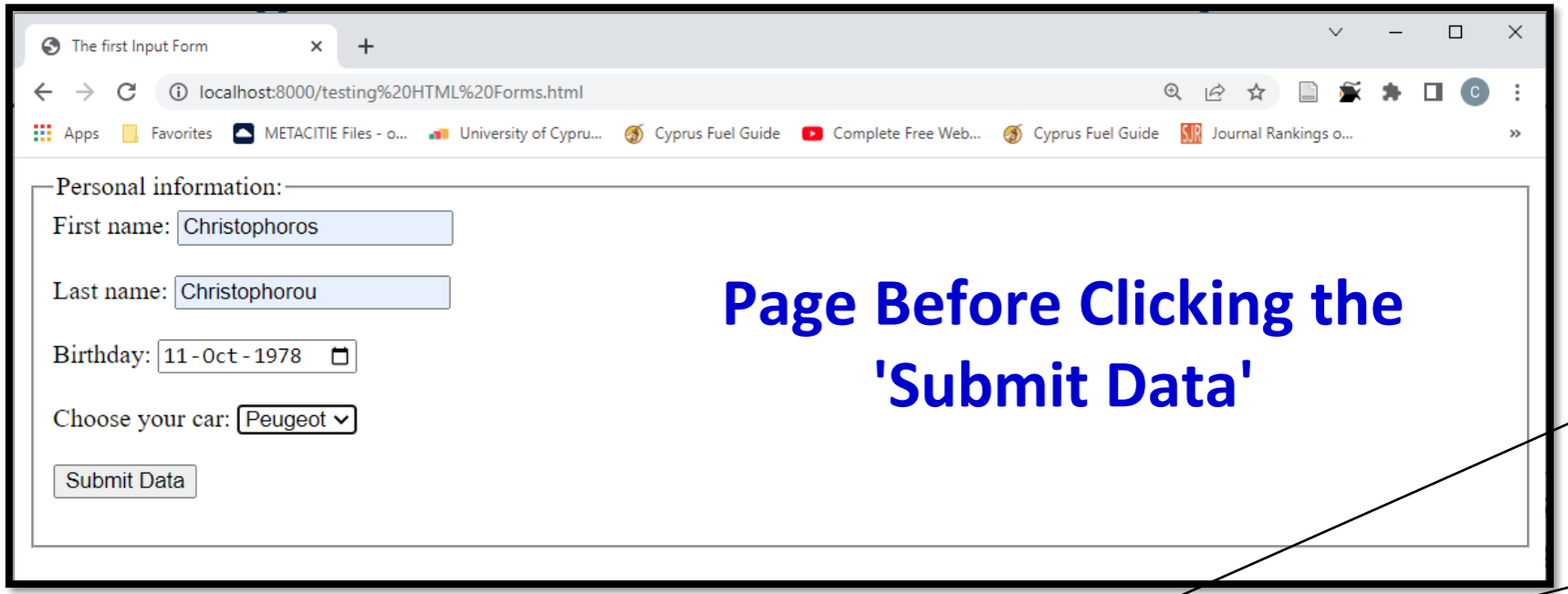
- **event.preventDefault()** basically **prevents 'submit' event to fire** and thus the page is **not refreshed**
- With this command **the default way** of submitting the form **is prevented.**
- For example, with **method="GET",** the form's data are **converted** into a **query string in name=value pairs (e.g., ?name1=value1&name2=value2)** and **added to the URL** (this is **the default way) – when event.preventDefault()** is used, this is **prevented.**

Accessing the form's data - using event.preventDefault()

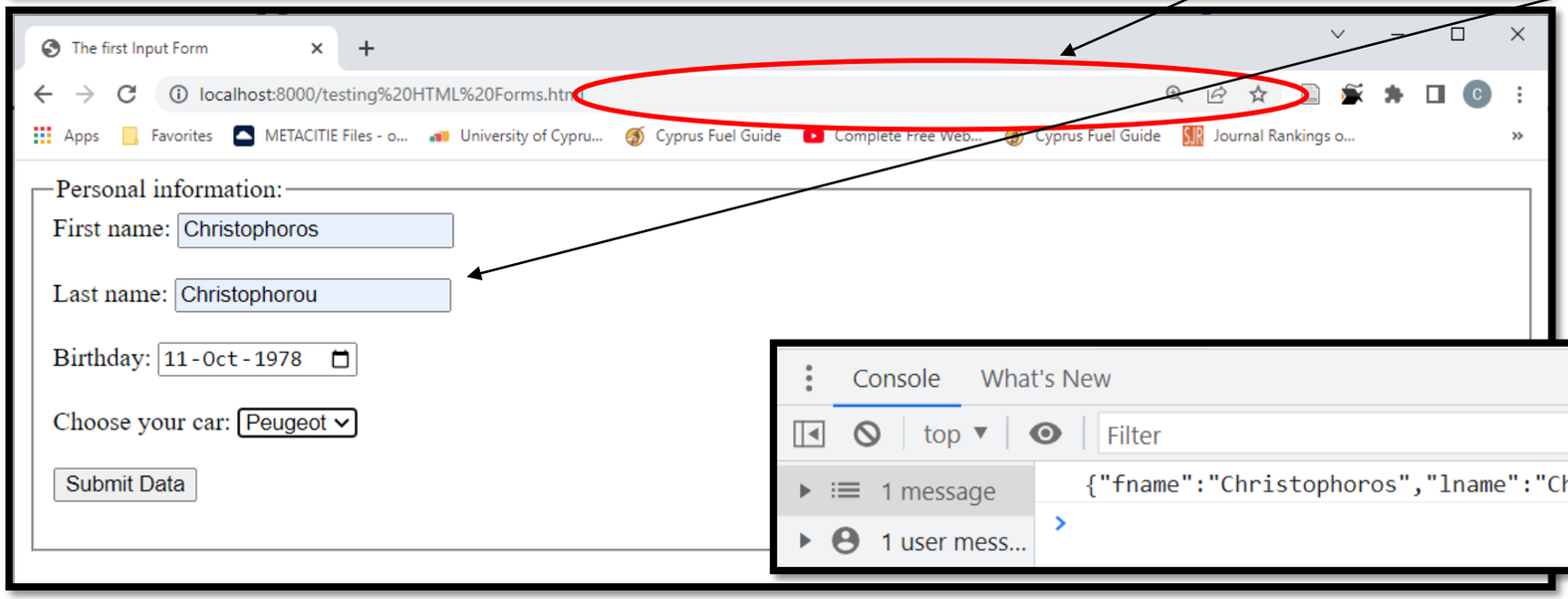
- We **normally prevent submit behavior if we want to**:
  - **Check data validation before submitting the form**
  - **Change values of our input fields** or
  - **Submit using AJAX calls**.

Note: We can also use <button type="button"> to create the button and **onclick event** call a **JavaScript function** to **submit the form's data**. In this case the **event.preventDefault()** command **will not be needed!**

Accessing the form's data - using event.preventDefault()



Page Before Clicking the 'Submit Data'



Page After Clicking the 'Submit Data' with event.preventDefault():

1. **Data is not added to the URL**
2. **The data in the form is not lost.**

Data is converted into JSON format and text remains in the Console

```javascript
// Here we add a 'submit' event listener on the form. After
// the 'Submit Data' is clicked a 'submit' event is fired and
// the getValues method is triggered.
var form = document.getElementById("form1");
form.addEventListener('submit', getValues);

// getValues method is the event listener function that receives
// a notification (an object that implements the Event Interface)
// when an event of the specified type (i.e., 'submit') occurs.
function getValues() {
    // Below we create an object to store our data. This object
    // can be later, using JSON.stringify, transformed to JSON format
     const person = {
        fname: document.getElementById("firstname").value,
        lname: document.getElementById("lastname").value,
        bdate: document.getElementById("birthday").value,
        car: document.getElementById("cars").value
     };

    // For debuggin purposes we write the data to the Console.
    // Data are written in JSON format.For this JSON.stringify is used
    console.log(JSON.stringify(person));
}
```

In this JavaScript code event.preventDefault() is not used…

Check in the next slide what will happen in this case…

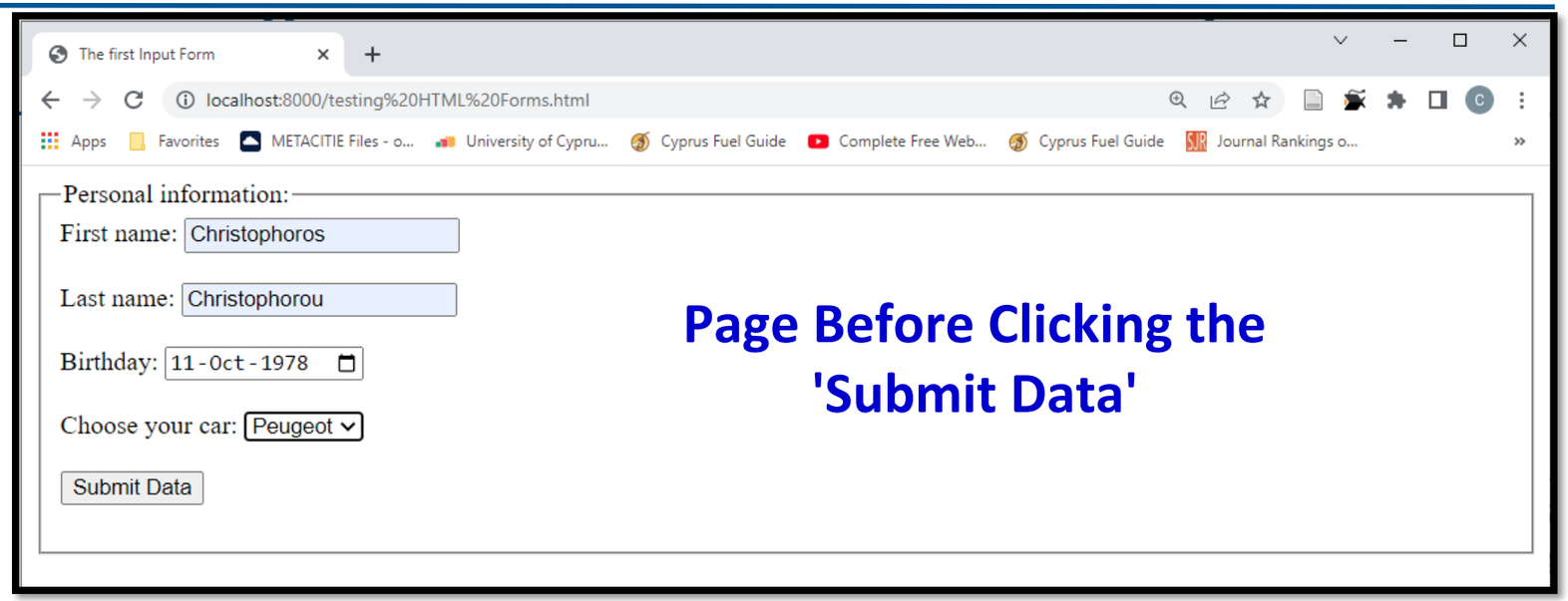
Note: This is the default way used when submitting data to the web server. In this way the page is AUTOMATICALLY redirected via an HTTP message to a php file on the server. This php file **should be provided** with the **form's action attribute.**

```html
<form id="form1" action="PHP/submit_data.php" method="GET">
```

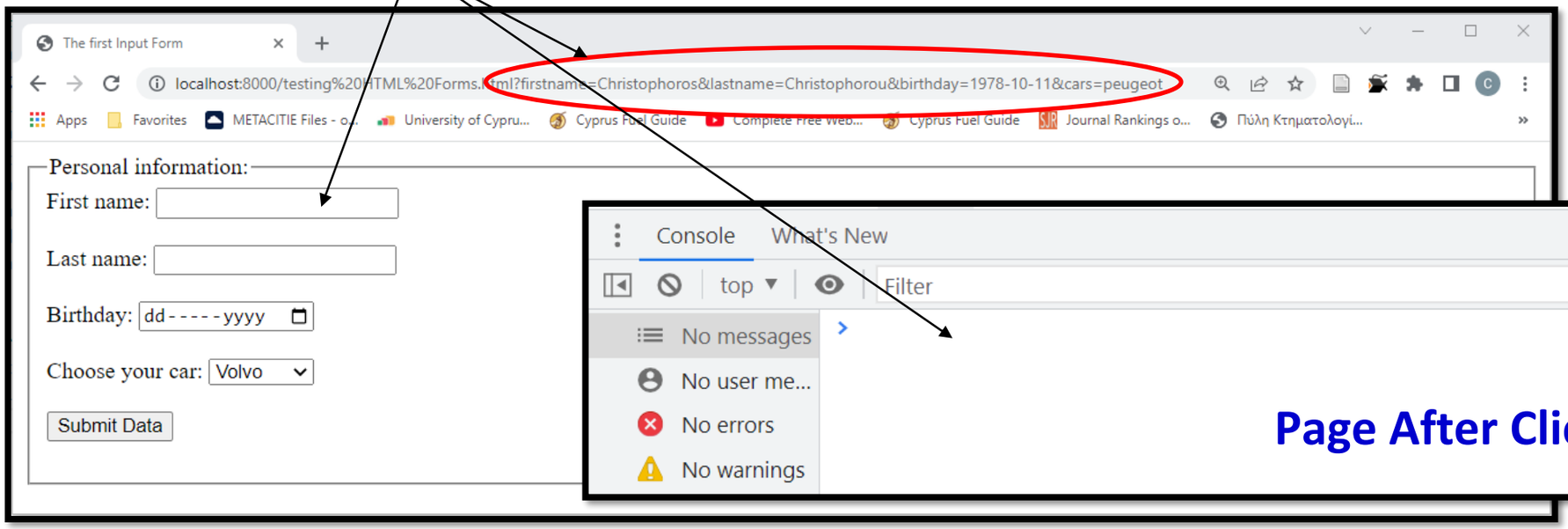
Accessing the form's data – with out event.preventDefault()

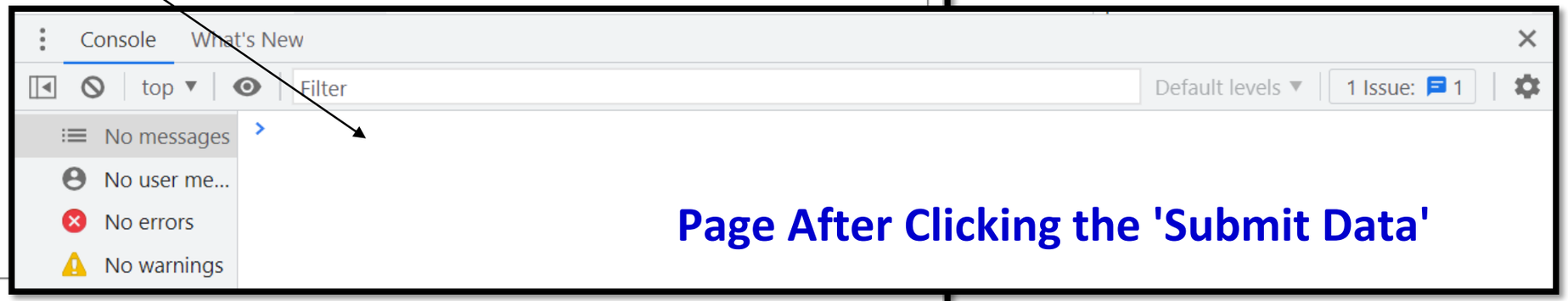
When '**Submit Data**' is clicked,

1. Form's data are **added to the URL after ?** in **key=value pairs**, separated with **&**
2. **The page** is **refreshed**, causing the **form's data** and the **JSON string** written in the console **to be erased**!



Page Before Clicking the 'Submit Data'





Page After Clicking the 'Submit Data'

Using AJAX – A Step by Step process

```
function handleResponse() {

    if (httpRequest.readyState === XMLHttpRequest.DONE) {
        // Everything is good, the response was received.

        if (httpRequest.status === 200) {
            // Perfect!

             var myObject = JSON.parse(httpRequest.responseText);
            // Assuming server sends a JSON format
            // Then do what ever you want with the data in myObject
        }
        else {
            // There was a problem with the request.
            // For example, the response may have a 404 (Not Found)
            // or 500 (Internal Server Error) response code.
        }

    } else {
        // Not ready yet.
    }
}
```

First, the function needs to check the **request's state**. If the state has the value of **XMLHttpRequest.DONE** (corresponding to 4), that means that **the server's full response was received** and to **continue processing it**.

Next, check the HTTP response status codes of the HTTP response.

In our example, we differentiate between a **successful** and **unsuccessful** AJAX call by checking for a 200 OK response code.

Using AJAX – A Step by Step process

```
function handleResponse() {

    if (httpRequest.readyState === XMLHttpRequest.DONE) {
        // Everything is good, the response was received.

        if (httpRequest.status === 200) {
            // Perfect!

             var myObject = JSON.parse(httpRequest.responseText);
            // Assuming server sends a JSON format
            // Then do what ever you want with the data in myObject
        }
        else {
            // There was a problem with the request.
            // For example, the response may have a 404 (Not Found)
            // or 500 (Internal Server Error) response code.
        }

    } else {
        // Not ready yet.
    }
}
```

After checking the state of the request and the HTTP status code of the response, **you can access the data the server sent**.

One option, that we will also use, to access that data, is using **httpRequest.responseText**.

This **.responseText** returns a string that contains the response to the **request as text** (i.e., in our case it will be **JSON format**), or **null** if the request was unsuccessful or has not yet been sent. This **.responseText** includes the **echo** of the PHP script.