Report project n.5

AJAX = **A**synchronous **J**avaScript **A**nd **X**ML.

AJAX is not a programming language it is a web development technique used to create fast and dynamic web pages by exchanging data with the web server without having to reload the entire page.
With AJAX, it is possible to create web pages that can update data dynamically without requiring a full page refresh.
AJAX requests can be initiated by JavaScript code on the client-side, which communicates with the web server asynchronously in the background. This means that the user can continue to interact with the page while the request is being processed in the background.
Here is an example of how AJAX can be used to fetch data from a server and update a web page dynamically:
(I created this example using the first project done in this subject laboratory)

```
<div id="demo">
<h2>The XMLHttpRequest Object</h2>
<button type="button" onclick="loadDoc()">Change Content</button>
</div>

<script>
function loadDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    document.getElementById("demo").innerHTML =
    this.responseText;
  }
  xhttp.open("GET", "index1.html");
  xhttp.send();
}
</script>
```

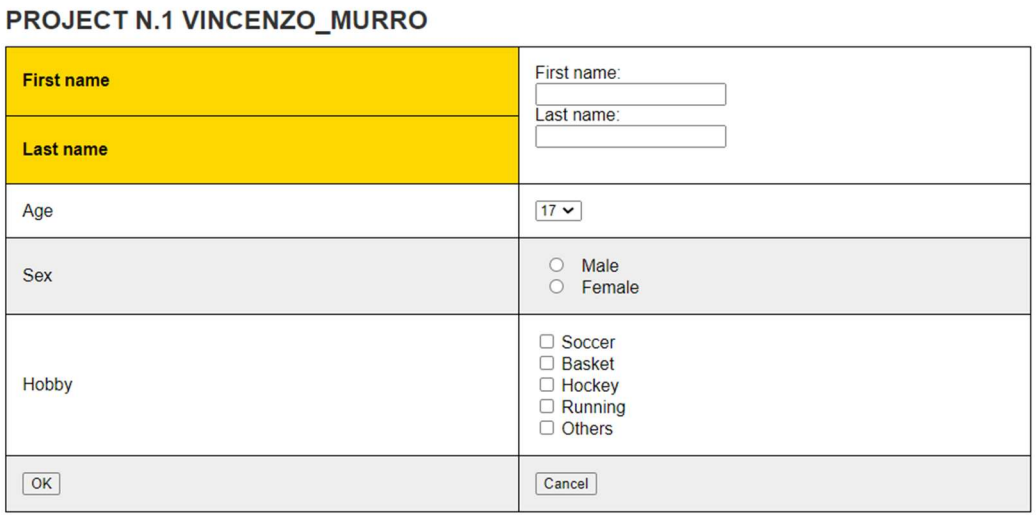
The code consists of an HTML section with a div element with an id of "demo" that contains an h2 element and a button element with an onclick event that triggers the loadDoc function. The loadDoc function creates a new XMLHttpRequest object and sets its onload property to a function that updates the innerHTML of the "demo" div element with the responseText from the HTTP request. The xhttp object then sends a GET request to the server to retrieve the

index1.html file. Once the response is received, the callback function is executed, and the content of the "demo" div element is updated with the responseText.

In summary, this code demonstrates how to use XMLHttpRequest to dynamically load content from an external file and update the webpage without reloading the whole page.

Result before clicking button:

Result after clicking button:



Task:

5. Featured Content Slider - Murro Vincenzo
http://www.dynamicdrive.com/dynamicindex17/featuredcontentslider.htm
Please utilize the Example of the website in order to build a menu to manage the Projects no. 1-4 we have done so far as multipage content (as it is done in the example). The menu will have as labels of the tabs (Project no.1, Project no.2 and so on…)

Code created for the task:

```
<div id="slider2" class="sliderwrapper">

</div>
<div id="paginate-slider2" class="pagination">
<a href="#" class="toc">Project n.1</a> <a href="#" class="toc
anotherclass">Project n.2</a> <a href="#" class="toc">Project n.3</a> <a href="#"
class="toc">Project n.4</a> <a href="#" class="prev" style="margin-left:
10px"><</a> <a href="#" class="next">></a>
</div>

<script type="text/javascript">

featuredcontentslider.init({
    id: "slider2",
    contentsource: ["ajax", "content.html"],
    toc: "markup",
    nextprev: ["Previous", "Next"],
    revealtype: "click",
    enablefade: [true, 0.2],
    autorotate: [false, 3000],
    onChange: function(previndex, curindex, contentdivs){

    }
})
</script>
```

This code is for a featured content slider, which displays a series of projects or other content items in a slideshow format. The HTML code contains two div elements, one with an id of "slider2" and the other with an id of "paginate-slider2". The first div will contain the slideshow, while the second will contain pagination links for navigating the slides.

The JavaScript code uses the featuredcontentslider library to initialize the slider with various options, specified using an object literal. These options include the ID of the main slider DIV, the source of the content (either inline or via AJAX), the type of table of contents (either "markup" or a list of labels), the labels for the "prev" and "next" links, the behavior of pagination links (either "click" or "mouseover"), whether to enable fading between slides, whether to enable automatic rotation of slides, and an onChange event handler that fires whenever the script changes slide.

In this case in order to practice with Ajax I defined another HTML file that contain the content of all the 4 project separated in 4 different div with the class ”contentdiv” as follow:

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
    <div class="contentdiv">···
        </div>/.contentdiv

    <div class="contentdiv">···
    </div>/.contentdiv

    <div class="contentdiv">···
    </div>/.contentdiv
    <div class="contentdiv">···
    </div>/.contentdiv

</body>
</html>
```

The content of every div class is hidden because contain every project code and it might have been too stretched to be displayed in this report.

As we could see previously I connected this file in this line of code:

```
contentsource: ["ajax", "content.html"]
```

In the contentslider.js file connected to the project it is possible to see ajax functionality, the most important library function is this:

```
ajaxconnect:function(setting){
    var page_request = false
```

```
    if (window.ActiveXObject){ //Test for support for ActiveXObject in IE first
(as XMLHttpRequest in IE7 is broken)
        try {
        page_request = new ActiveXObject("Msxml2.XMLHTTP")
        }
        catch (e){
            try{
            page_request = new ActiveXObject("Microsoft.XMLHTTP")
            }
            catch (e){}
        }
    }
    else if (window.XMLHttpRequest) // if Mozilla, Safari etc
        page_request = new XMLHttpRequest()
    else
        return false
    var pageurl=setting.contentsource[1]
    page_request.onreadystatechange=function(){
        featuredcontentslider.ajaxpopulate(page_request, setting)
    }
    document.getElementById(setting.id).innerHTML=this.ajaxloadingmsg
    var bustcache=(!this.bustajaxcache)? "" : (pageurl.indexOf("?")!=-1)? "&"+new
Date().getTime() : "?"+new Date().getTime()
    page_request.open('GET', pageurl+bustcache, true)
    page_request.send(null)
}
```

This code defines a JavaScript function called ajaxconnect that takes in a setting object as its parameter. The purpose of this function is to make an AJAX request to retrieve content from a specified URL and populate it into a specified element on a web page.

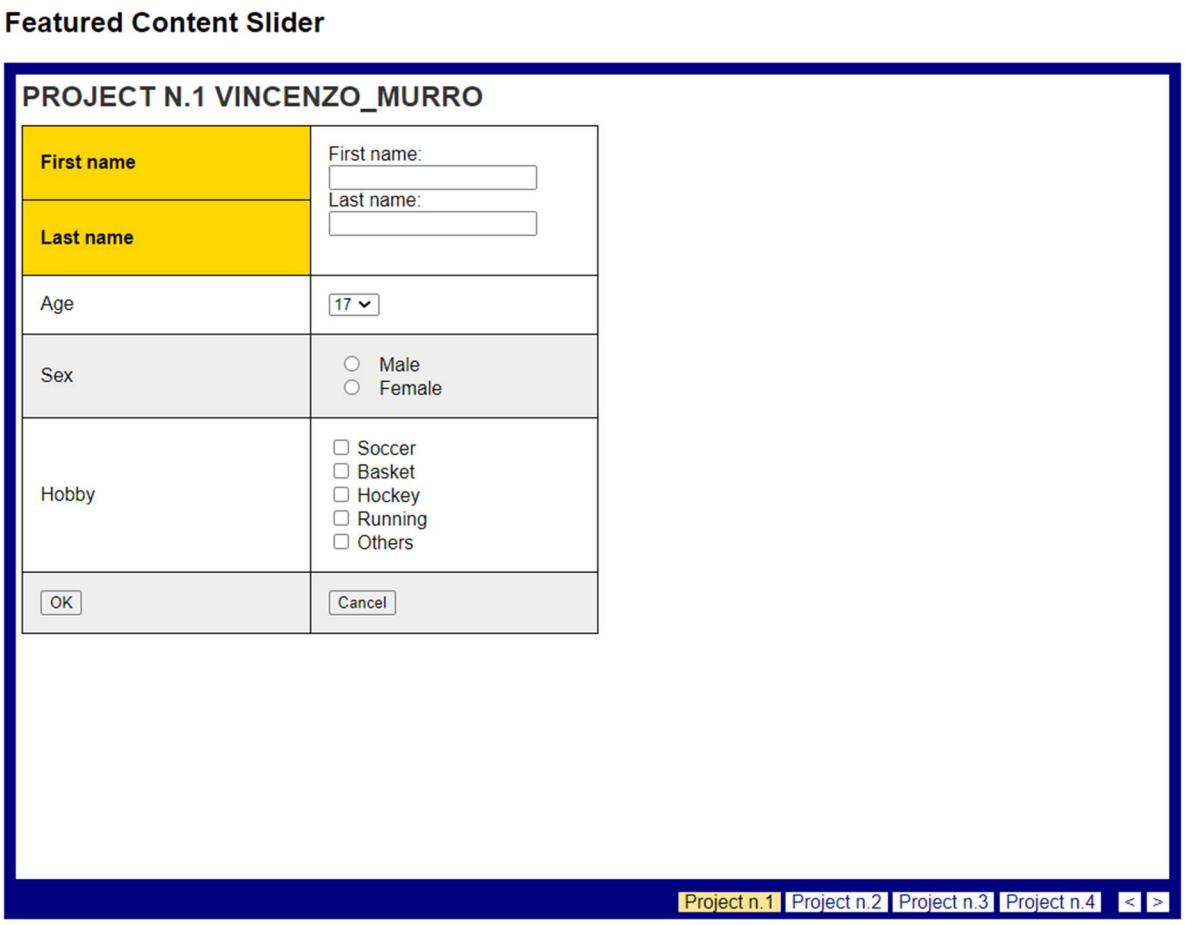
The function checks if the user's browser supports the ActiveXObject or XMLHttpRequest objects for making AJAX requests, creates an instance of the appropriate object, and assigns it to a variable called page_request. It sets a callback function to be called whenever the readyState property of page_request changes, and then uses the open and send methods of page_request to initiate the AJAX request. The retrieved content is then populated into the specified element on the web page using a separate function called ajaxpopulate.

```
ajaxpopulate:function(page_request, setting){
    if (page_request.readyState == 4 && (page_request.status==200 ||
window.location.href.indexOf("http")==-1)){
        document.getElementById(setting.id).innerHTML=page_request.responseText
```

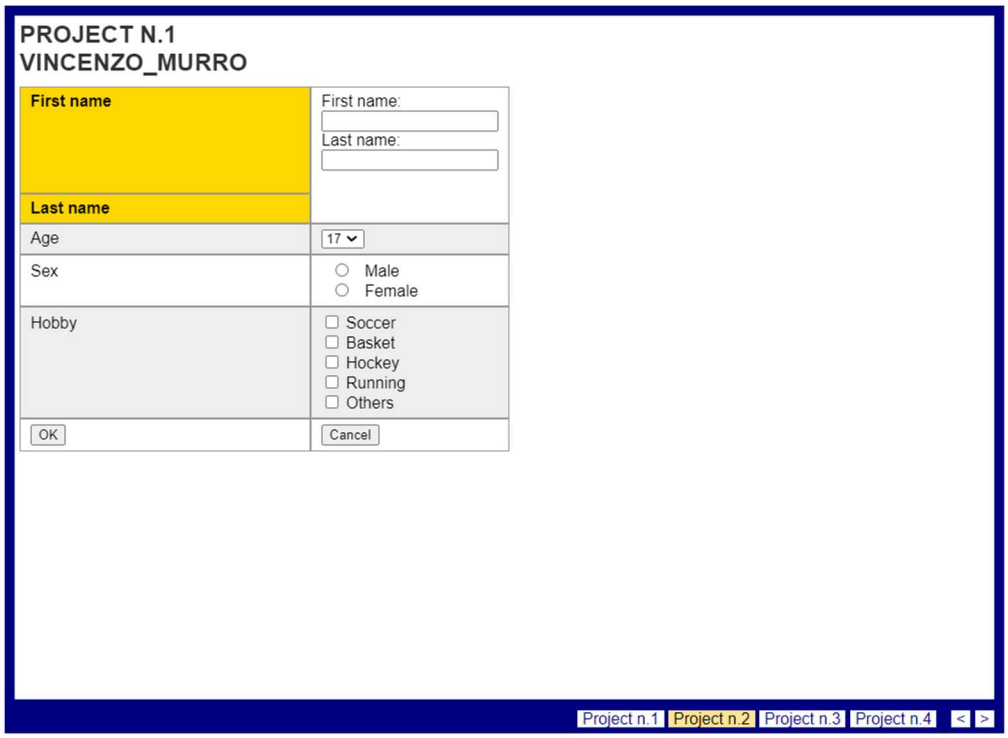
```
        this.buildpaginate(setting)
    }
}
```

The function first checks if the readyState property of page_request is equal to 4 (which indicates that the request has been completed) and if the status property of page_request is equal to 200 (which indicates that the request was successful) or if the current URL does not contain "http". If both of these conditions are true, the function sets the innerHTML property of the element with the ID specified in setting.id to the responseText property of page_request, which contains the retrieved content. It then calls the buildpaginate function to create pagination links for the retrieved content.

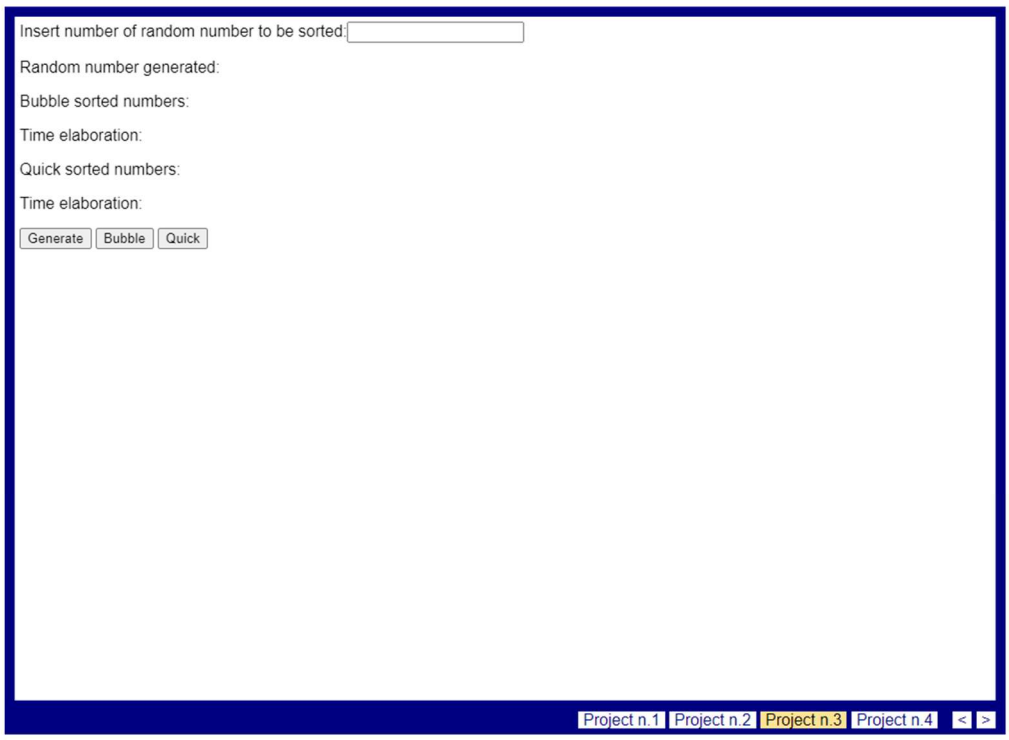
Final results of the code:



Featured Content Slider



Featured Content Slider



Featured Content Slider