**Assignment 3**

**Roll no: 31164 Date: 26/05/21**

**TITLE**: Perform validations using JavaScript / JQuery

**PROBLEM STATEMENT:** Perform validations using JavaScript / JQuery on one of the applications from the given list :

1. Online pizza order application
2. Student information system for training & placement department
3. Leave management application
4. Blogging platform
5. Meeting room booking application
6. Exam cell automation application

**OBJECTIVE:**

* To understand in depth working of JavaScript and Jquery
* To use JavaScript and Jquery for providing validations, animation and effects to any web application.

**S/W PACKAGES AND HARDWARE APPARATUS USED:** Operating System open source Fedora 20 Networked computer with internet access

Editor : IDE : Netbeans 8.1

Web browser Mozilla Firefox, Google Chrome

**THEORY:**

1. **What is JavaScript?**

JavaScript was initially created to “make webpages alive”.

The programs in this language are called scripts. They can be written right in the HTML and execute automatically as the page loads. Scripts are provided and executed as a plain text. They don‘t need a special preparation or a compilation to run. In this aspect, JavaScript is very different from another language called [Java.](http://en.wikipedia.org/wiki/Java)

1. **Why JavaScript?**

When JavaScript was created, it initially had another name: ―LiveScript. But Java language was very popular at that time, so it was decided that positioning a new language as a ―younger brother of Java would help. But as it evolved, JavaScript became a fully independent language, with its own specification called [ECMAScript,](http://en.wikipedia.org/wiki/ECMAScript) and now it has no relation to Java at all. At present, JavaScript can execute not only in the browser, but also on the server, or actually on any device where there exists a special program called [the JavaScript engine.](https://en.wikipedia.org/wiki/JavaScript_engine) The browser has an embedded engine, sometimes it‘s also called a ―JavaScript virtual machine.

Different engines have different ―codenames, for example:

* [V8](https://en.wikipedia.org/wiki/V8_(JavaScript_engine)) – in Chrome and Opera.
* [SpiderMonkey](https://en.wikipedia.org/wiki/SpiderMonkey) – in Firefox.
* There are other codenames like ―Trident, ―Chakra for different versions of IE, ―ChakraCore for Microsoft Edge, ―Nitro and ―SquirrelFish for Safari etc.

The terms above are good to remember, because they are used in developer articles on the internet.

1. **How engines work?**

Engines are complicated. But the basics are definable.

* The engine (embedded if it‘s a browser) reads (―parses) the script.
* Then it converts (―compiles) the script to the machine language.
* And then the machine code runs, pretty fast.

The engine applies optimization on every stage of the process. It even watches the compiled script as it runs, analyzes the data that flows through it and applies optimization to the machine code based on that knowledge. At the end, scripts are quite fast.

1. **What can in-browser JavaScript do?**

The modern JavaScript is a ―safe programming language. It does not provide low-level access to memory or CPU, because it was initially created for browsers which do not require it. The capabilities greatly depend on the environment that runs JavaScript. For instance, [Node.JS](https://wikipedia.org/wiki/Node.js) supports functions that allow JavaScript to read/write arbitrary files, perform network requests etc. In-browser JavaScript can do everything related to webpage manipulation, interaction with the user and the webserver.

For instance, in-browser JavaScript is able to:

* Add new HTML to the page, change the existing content, modify styles.
* React to user actions, run on mouse clicks, pointer movements, key presses.
* Send requests over the network to remote servers, download and upload files (so-called [AJAX](https://en.wikipedia.org/wiki/Ajax_(programming)) and [COMET](https://en.wikipedia.org/wiki/Comet_(programming)) technologies).
* Get and set cookies, ask questions to the visitor, show messages.
* Remember the data on the client-side (―local storage).

1. **What CAN’T in-browser JavaScript do?**

JavaScript‘s abilities in the browser are limited for the sake of the user‘s safety. The aim is to prevent an evil webpage from accessing private information or harming the user‘s data. The examples of such restrictions are:

* JavaScript on a webpage may not read/write arbitrary files on the hard disk, copy them or execute programs. It has no direct access to OS system functions.

Modern browsers allow it to work with files, but the access is limited and only provided if the user does certain actions, like ―dropping‖ a file into a browser window or selecting it via an <input> tag.

There are ways to interact with camera/microphone and other devices, but they require a user‘s explicit permission. So a JavaScript-enabled page may not sneakily enable a web-camera, observe the surroundings and send the information to the [NSA.](https://en.wikipedia.org/wiki/National_Security_Agency)

* Different tabs/windows generally do not know about each other. Sometimes they do, for example when one window uses JavaScript to open the other one. But even in this case, JavaScript from one page may not access the other if they come from different sites (from a different domain, protocol or port).

This is called the ―Same Origin Policy. To work around that, both pages must contain a special JavaScript code that handles data exchange.

The limitation is again for user‘s safety. A page from http://anysite.com which a user has opened must not be able to access another browser tab with the URL http://gmail.com and steal information from there.

* JavaScript can easily communicate over the net to the server where the current page came from. But its ability to receive data from other sites/domains is crippled. Though possible, it requires explicit agreement (expressed in HTTP headers) from the remote side. Once again, that‘s safety limitations.

1. **What makes JavaScript unique?**

* Full integration with HTML/CSS.
* Simple things done simply.
* Supported by all major browsers and enabled by default.

Combined, these three things exist only in JavaScript and no other browser technology. That‘s what makes JavaScript unique. That‘s why it‘s the most widespread tool to create browser interfaces. While planning to learn a new technology, it‘s beneficial to check its perspectives.

### Limitations of JavaScript

We cannot treat JavaScript as a full-fledged programming language as it lacks the following important features −

* Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
* JavaScript cannot be used for networking applications because there is no such support available.
* JavaScript doesn't have any multithreading or multiprocessor capabilities.

Once again, JavaScript is a lightweight, interpreted programming language that allows you to build interactivity into otherwise static HTML pages.

### Languages “over” JavaScript

The syntax of JavaScript does not suit everyone‘s needs. Different people want different features. That‘s to be expected, because projects and requirements are different for everyone.

So recently a plethora of new languages appeared, which are transpiled (converted) to JavaScript before they run in the browser.

Modern tools make the transpilation very fast and transparent, actually allowing developers to code in another language and autoconverting it ―under the hood‖.

Examples of such languages:

* [CoffeeScript](http://coffeescript.org/) is a ―syntactic sugar‖ for JavaScript, it introduces shorter syntax, allowing to write more precise and clear code. Usually Ruby devs like it.
* [TypeScript](http://www.typescriptlang.org/) is concentrated on adding ―strict data typing‖, to simplify development and support of complex systems. It is developed by Microsoft.
* [Dart](https://www.dartlang.org/) is a standalone language that has its own engine that runs in non-browser environments (like mobile apps). It was initially offered by Google as a replacement for JavaScript, but as of now, browsers require it to be transpiled to JavaScript just like the ones above.

There are more. Of course even if we use one of those languages, we should also know JavaScript, to really understand what we‘re doing.

**Login Form Validation using JavaScript** JavaScript code:

|  |
| --- |
| <script language = "JavaScript"> function validate() {  var username = document.getElementById("username").value;  var password = document.getElementById("password").value; if (username == null || username == "")  {  alert("Please enter the username.");  return false;  }  if (password == null || password == "") { alert("Please enter the password."); return false;  }  alert('Login successful');  }  </script> |

HTML code:

|  |
| --- |
| <html>  <body>  <form id="form1" runat="server">  <div class="container">  <div class="main">  <h2> Javascript Login Form Validation</h2>  <form id="form\_id" method="post" name="myform">  <label> User Name :</label>  <input type="text" name="username" id="username" /> <br><br>  <label> Password :</label>  <input type="password" name="password" id="password" />  <input type="button" value="Login" id="submit" onclick="validate();" /> </form>  </div> </div> </form>  </html> </body> |

### Summary of JavaScript

* JavaScript was initially created as a browser-only language, but now it is used in many other environments as well.
* At this moment, JavaScript has a unique position as the most widely-adopted browser language with full integration with HTML/CSS.
* There are many languages that get ―transpiled to JavaScript and provide certain features. It is recommended to take a look at them, at least briefly, after mastering JavaScript.

## JQUERY

### About JQuery

JQuery is just a JavaScript library. All the power of jQuery is accessed via JavaScript, so having a strong grasp of JavaScript is essential for understanding, structuring, and debugging your code. While working with jQuery regularly can, over time, improve your proficiency with JavaScript, it can be hard to get started writing jQuery without a working knowledge of JavaScript's built-in constructs and syntax.

1. **How JQuery works?**

This is a basic tutorial, designed to help you get started using jQuery. If you don't have a test page setup yet, start by creating the following HTML page:

<!doctype html>

<html>

<head>

<meta charset="utf-8">

<title>Demo</title>

</head>

<body>

<a href="http://jquery.com/">jQuery</a>

<script src="jquery.js"></script>

<script>

// Your code goes here.

</script>

</body>

</html>

The src attribute in the <script> element must point to a copy of jQuery. Download a copy of jQuery from the [Downloading jQuery](http://jquery.com/download/) page and store the jquery.js file in the same directory as your HTML file.

Note: When you download jQuery, the file name may contain a version number, e.g., jquery-x.y.z.js. Make sure to either rename this file to jquery.js or update the src attribute of the <script> element to match the file name.

### Launching Code on Document Ready:

To ensure that their code runs after the browser finishes loading the document, many JavaScript programmers wrap their code in an onload function:

|  |
| --- |
| window.onload = function() {  alert( "welcome" ); }; |

Unfortunately, the code doesn't run until all images are finished downloading, including banner ads. To run code as soon as the document is ready to be manipulated, jQuery has a statement known as the [ready event:](http://api.jquery.com/ready/)

$( document ).ready(function() {

// Your code here.

});

Note: The jQuery library exposes its methods and properties via two properties of the window object called jQuery and $. $ is simply an alias for jQuery and it's often employed because it's shorter and faster to write.

For example, inside the ready event, you can add a click handler to the link:

$( document ).ready(function() {

$( "a" ).click(function( event ) {

alert( "Thanks for visiting!" );

});

});

Copy the above jQuery code into your HTML file where it says // Your code goes here. Then, save your HTML file and reload the test page in your browser. Clicking the link should now first display an alert pop-up, then continue with the default behavior of navigating to [http://jquery.com.](http://jquery.com/)

For click and most other [events,](http://api.jquery.com/category/events/) you can prevent the default behavior by calling event.preventDefault() in the event handler:

$( document ).ready(function() {

$( "a" ).click(function( event ) {

alert( "As you can see, the link no longer took you to jquery.com" );

event.preventDefault();

});

});

Try replacing your first snippet of jQuery code, which you previously copied in to your HTML file, with the one above. Save the HTML file again and reload to try it out.

### Complete Example

The following example illustrates the click handling code discussed above, embedded directly in the HTML <body>. Note that in practice, it is usually better to place your code in a separate JS file and load it on the page with a <script> element's src attribute.

|  |
| --- |
| <!doctype html>  <html>  <head>  <meta charset="utf-8">  <title>Demo</title>  </head>  <body>  <a href="http://jquery.com/">jQuery</a>  <script src="jquery.js"></script>  <script>    $( document ).ready(function() { $( "a" ).click(function( event ) {  alert( "The link will no longer take you to jquery.com" ); event.preventDefault();  });  });    </script>  </body>  </html> |

### Callbacks and Functions

Unlike many other programming languages, JavaScript enables you to freely pass functions around to be executed at a later time. A callback is a function that is passed as an argument to another function and is executed after its parent function has completed. Callbacks are special because they patiently wait to execute until their parent finishes. Meanwhile, the browser can be executing other functions or doing all sorts of other work.

To use callbacks, it is important to know how to pass them into their parent function.

* **Callback without Arguments**

If a callback has no arguments, you can pass it in like this:

$.get( "myhtmlpage.html", myCallBack );

When [$.get()](http://api.jquery.com/jQuery.get/) finishes getting the page myhtmlpage.html, it executes the myCallBack() function. Note: The second parameter here is simply the function name (but not as a string, and without parentheses).

* **Callback with Arguments**

Executing callbacks with arguments can be tricky.

This code example will not work:

$.get( "myhtmlpage.html", myCallBack( param1, param2 ) );

The reason this fails is that the code executes myCallBack( param1, param2 ) immediately and then passes myCallBack()'s return value as the second parameter to $.get(). We actually want to pass the function myCallBack(), not myCallBack( param1, param2 )'s return value (which might or might not be a function).

To defer executing myCallBack() with its parameters, you can use an anonymous function as a wrapper. Note the use of function() {. The anonymous function does exactly one thing: calls myCallBack(), with the values of param1 and param2.

$.get( "myhtmlpage.html", function() {

myCallBack( param1, param2 );

});

When $.get() finishes getting the page myhtmlpage.html, it executes the anonymous function, which executes myCallBack( param1, param2 ).

## Login form validations using JQuery

<!DOCTYPE html>

<html lang="en">

<head> <meta charset="utf-8"> <title>jQuery validation login basic example</title>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>

<meta name="viewport" content="width=device-width"/>

<script src="http://code.jquery.com/jquery-1.11.0.min.js"></script>

<script src="http://ajax.aspnetcdn.com/ajax/jquery.validate/1.13.0/jquery.validate.min.js"></script> <script>

//Configuration

var minUserLen = 5, maxUserLen = 30;

var minPassLen = 8, maxPassLen = 4096;

var usernameMsg = "Username must be between " + minUserLen + " and " + maxUserLen + " characters, inclusive.";

var passwordMsg = "Password must be between " + minPassLen + " and " + maxPassLen + " characters, inclusive."; jQuery.validator.setDefaults({ debug: true, //Avoids form submit. Comment when in production.

success: "valid", submitHandler: function() {

alert("Success! The form was pretend-submitted!");

} });

$(document).ready(function() {

// validate signup form on keyup and submit

$("#signupForm").validate({ rules: { username: { required: true, minlength: minUserLen,

maxlength: maxUserLen

}, password: { required: true, minlength: minPassLen,

maxlength: maxPassLen

}, }, messages: { username: { required: "Username required", minlength: usernameMsg, maxlength: usernameMsg

}, password: { required: "Password required", minlength: passwordMsg, maxlength: passwordMsg

} } }); });

</script>

</head>

<body>

<p>Validating a login form with the <a href="http://jqueryvalidation.org/">JQuery Validation</a> plugin. The fields are first validated when the form either <a href="https://developer.mozilla.org/en-

US/docs/Mozilla/Tech/XUL/Attribute/textbox.onblur"><code>onBlur</code></a> or when submitted. Any fields found to have an error are then validated <a href="https://developer.mozilla.org/en-

US/docs/Web/API/Window.onkeyup"><code>onkeyup</code></a>. This is a whittled-down version of top example on the <a href="http://jqueryvalidation.org/files/demo/">main JQuery example page</a>.</p>

<form id="signupForm">

<fieldset>

<legend>Fake login please</legend>

<p>

<label for="username">Username</label>

<input id="username" name="username" type="text">

</p>

<p>

<label for="password">Password</label>

<input id="password" name="password" type="password">

</p> <p>

<input class="submit" type="submit" value="Submit">

</p>

</fieldset>

</form>

</body>

</html>

**Conclusion:** We learned implementation of validation of forms using Javascript and JQuery.