WEB PROGRAMMING AND APPLICATIONS (Tutorial 7)

Table of Contents

jQuery

What's jQuery

jQuery is a fast, small, and feature-rich JavaScript library. It simplifies various tasks involved in client-side scripting of HTML, such as HTML document traversal and manipulation, event handling, animation, and AJAX (asynchronous JavaScript and XML) interactions.

- DOM Manipulation: jQuery provides a simple and efficient way to traverse and manipulate the HTML Document Object Model (DOM). It allows developers to easily select elements, apply CSS styles, modify content, and handle events.
- Event Handling: jQuery simplifies event handling by providing methods like **click(), hover(), keyup(),** etc., which make it easier to attach event listeners to DOM elements.
- AJAX: jQuery simplifies AJAX requests with its **$.ajax()** function, making it easier to send and receive data from a web server asynchronously without reloading the entire page.
- Cross-Browser Compatibility: jQuery abstracts away many of the differences between JavaScript implementations across different browsers, making it easier to write code that works consistently across various platforms.
- Animation: jQuery provides built-in animation effects and methods to create smooth animations on web pages, such as **fadeIn(), fadeOut(), slideDown(), slideUp(),** etc.
- Extensibility: jQuery is highly extensible, allowing developers to create custom plugins and extensions to enhance its functionality or solve specific problems.

Add jQuery to web pages

To add jQuery to web pages, you have a few options:

Download jQuery and Host Locally:

- Download the jQuery library from the official jQuery website.
- Save the downloaded jQuery file (usually named jquery-x.y.z.js or jquery-x.y.z.min.js, where x.y.z is the version number) to your project directory.
- Reference the jQuery file in your HTML pages using a <script> tag:

```
<script src="path/to/jquery-x.y.z.js"></script>
```

Use a CDN (Content Delivery Network):

- Link to a hosted version of jQuery from a CDN provider. This approach is often used because it can improve page load times by leveraging caching and distributing the file from servers geographically closer to the user.
- Here's an example of how to include jQuery from the Google CDN:

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
```

jQuery Syntax

jQuery syntax is designed to be simple and intuitive, allowing developers to perform various tasks like selecting elements, manipulating the DOM, handling events, and making AJAX requests with ease. Here's an overview of common jQuery syntax:

Selecting Elements

Use the $() function to select elements. You can pass in a CSS selector or a DOM element.

```
$(document) // Selects the entire document
$('#myElement') // Selects an element with id="myElement"
$('.myClass') // Selects elements with class="myClass"
```

DOM Manipulation

jQuery provides methods for manipulating the DOM, such as adding or removing elements, changing content, or modifying attributes.

```
// Sets the text content of an element
$('#myElement').text('Hello, world!');

// Appends new content to an element
$('#myElement').append('<p>New paragraph</p>');

// Sets an attribute value
$('#myElement').attr('title', 'New title');
```

Event Handling

jQuery simplifies event handling with methods like .on(), .click(), .keydown(), etc.

```
$('#myButton').click(function() {
    console.log('Button clicked');
});
```

Iterating Over Collections

jQuery makes it easy to iterate over collections of elements using methods like .each().

```
$('li').each(function(index, element) {
    console.log(index + ': ' + $(element).text());
});
```

AJAX Requests

jQuery provides shorthand methods for making AJAX requests, such as $.ajax(), $.get(), and $.post().

```
$.ajax({
    url: 'example.php',
    method: 'GET',
    success: function(data) {
        console.log('Data received:', data);
    },
    error: function(xhr, status, error) {
        console.error('Error:', error);
    }
});
```

Chaining Methods

jQuery allows you to chain multiple methods together, making your code concise and readable.

```
$('#myElement').addClass('highlight').fadeOut('slow');
```

PHP session

In PHP, sessions provide a way to store information across multiple pages during a user's visit to a website. Sessions are commonly used to store user-specific data such as login information, shopping cart contents, user preferences, etc.

Here's how sessions work in PHP:

1. **Session Start**: The session starts when the **session_start()** function is called. This function must be called before any output is sent to the browser, typically at the beginning of a PHP script.

   ```php
   <?php
   session_start();
   ?>
   ```

2. **Session Variables**: Once the session is started, you can store data in session variables. Session variables are accessible across different pages of your website as long as the session is active. You can set session variables using the **$_SESSION** superglobal array.

   ```php
   <?php
   // Store data in session variable
   $_SESSION['username'] = 'john_doe';
   ?>
   ```

3. **Retrieve Session Data**: You can retrieve session data from session variables using the **$_SESSION** superglobal array.

   ```php
   <?php
   // Retrieve data from session variable
   $username = $_SESSION['username'];
   echo "Welcome back, $username!";
   ?>
   ```

4. **Destroying a Session**: When a user logs out or the session needs to be destroyed for any reason, you can call the **session_destroy()** function. This function will unset all session variables and end the session.

   ```php
   <?php
   // Destroy the session
   session_destroy();
   ?>
   ```

Sessions are typically managed using cookies. When a session is started, PHP generates a unique session ID for the user, which is stored as a cookie on the user's browser. This session ID is then used to identify the user's session on subsequent requests, allowing PHP to retrieve the session data associated with that session ID.

It's important to note that session data is stored on the server, not on the user's browser. This makes sessions more secure than using cookies alone for storing sensitive data, as session data cannot be easily manipulated by the user. However, it also means that session data can consume server resources, so it's important to use sessions judiciously and destroy them when they are no longer needed.

PHP Files

Common file functions

file_exists

The file_exists() function in PHP is used to check whether a file or directory exists. It returns true if the file or directory exists, and false otherwise.

Here's an example:

```
$file = 'example.txt';

if (file_exists($file)) {
    echo "The file $file exists";
} else {
    echo "The file $file does not exist";
}
```

In this example, if `example.txt` exists in the current directory, it will output "The file example.txt exists". Otherwise, it will output "The file example.txt does not exist".

basename

The basename() function in PHP is used to return the base name of a file path. The base name is the last component of the path, after the last directory separator.

Here's the syntax:

```
string basename ( string $path [, string $suffix ] )
```

- $path: The path to get the base name from.
- $suffix (optional): If provided, this suffix will be stripped from the end of the base name.

Here's an example:

```
$path = "/home/user/example.txt";
echo basename($path); // Output: example.txt

$path = "/home/user/example.txt.zip";
```

```php
echo basename($path, ".zip"); // Output: example.txt
```

mkdir

The mkdir() function in PHP is used to create a new directory (folder).

Here's the syntax:

```php
bool mkdir ( string $directory [, int $mode = 0777 [, bool $recursive = FALSE ]] )
```

- **$directory**: The directory path that you want to create.
- **$mode** (optional): The mode of the newly created directory. Default is 0777 (octal notation).
- **$recursive** (optional): If set to TRUE, the function will create directories recursively. Default is FALSE.

Example:

```php
$dir = "/path/to/directory";

// Create a directory
if (!mkdir($dir, 0777)) {
    die("Failed to create directory");
} else {
    echo "Directory created successfully";
}
```

pathinfo

The pathinfo() function in PHP is used to return information about a file path. It parses the given path and returns an associative array containing information such as directory name, basename, extension, and filename without extension.

Here's the syntax of the pathinfo() function:

```php
pathinfo ( string $path [, int $options = PATHINFO_DIRNAME | PATHINFO_BASENAME
| PATHINFO_EXTENSION | PATHINFO_FILENAME ] ) : array|string|null
```

Parameters:

- **$path**: The file path for which information is to be returned.
- **$options** (optional): A bitmask of one or more of the following constants: **PATHINFO_DIRNAME**, **PATHINFO_BASENAME**, **PATHINFO_EXTENSION**, and **PATHINFO_FILENAME**. You can combine these constants using the bitwise OR (**|**) operator. By default, all components are returned.

Return Value:

- If the **$options** parameter is not provided, an associative array containing information about the path is returned.
- If a specific component is requested using the **$options** parameter, a string containing the requested component is returned.
- If the path is invalid or empty, **NULL** is returned.

Here's an example usage of **pathinfo()**:

```
$path = '/path/to/example.txt';

// Get all components of the path
$pathInfo = pathinfo($path);
print_r($pathInfo);

// Get the filename without extension
$filenameWithoutExtension = pathinfo($path, PATHINFO_FILENAME);
echo "Filename without extension: $filenameWithoutExtension\n";
```

Output:

```
Array
(
    [dirname] => /path/to
    [basename] => example.txt
    [extension] => txt
    [filename] => example
)
Filename without extension: example
```

$_FILES

$_FILES is a superglobal array in PHP that is used to collect file upload data submitted through an HTML form with the **enctype="multipart/form-data"** attribute. When a file is uploaded via a form, PHP populates the **$_FILES** array with information about the uploaded file.

The **$_FILES** array has the following structure:

1. **name**: The original name of the file on the client's machine.
2. **type**: The MIME type of the file (e.g., "image/jpeg").
3. **tmp_name**: The temporary filename of the file on the server. This is the filename where the uploaded file is stored on the server before it is moved to its permanent location.
4. **error**: The error code associated with the file upload (if any). A value of **UPLOAD_ERR_OK** means there is no error.
5. **size**: The size of the uploaded file in bytes.

Here's a basic example of how to handle file uploads using **$_FILES**:

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Check if there was no file upload error
    if ($_FILES["file"]["error"] == UPLOAD_ERR_OK) {
        // Move the uploaded file to a permanent location
        $targetDir = "uploads/";
        $targetFile = $targetDir . basename($_FILES["file"]["name"]);
        if (move_uploaded_file($_FILES["file"]["tmp_name"], $targetFile)) {
            echo "The file " . basename($_FILES["file"]["name"]) . " has been
uploaded.";
        } else {
            echo "Sorry, there was an error uploading your file.";
        }
    } else {
        echo "Error: " . $_FILES["file"]["error"];
    }
}
```

File management application

Login

```php
session_start();

if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $username = $_POST['username'];
    $password = $_POST['password'];
    if ($username === $mock_username && $password === $mock_password) {
        $_SESSION['username'] = $username;
        header("Location: home.php");
        exit;
    } else {
        $error = "Invalid username or password";
    }
}
```

Check login

```php
session_start();

if (!isset($_SESSION['username'])) {
   header("Location: login.php");
   exit;
}

$username = $_SESSION['username'];
```

Logout

Back-end (PHP)

```php
session_start();

$_SESSION = array();

session_destroy();
```

Front-end (JS)

```
function startCountdown(seconds) {
  var countdownElement = document.getElementById('countdown');
  var loginBtn = document.getElementById('loginBtn');

  var countdown = seconds;
  countdownElement.textContent = countdown;

  var timer = setInterval(function () {
    countdown--;
    countdownElement.textContent = countdown;

    if (countdown <= 0) {
      clearInterval(timer);
      loginBtn.click();
    }
  }, 1000);
}

$(document).ready(function () {
  startCountdown(10);

  $('#loginBtn').click(function () {
    window.location.href = 'login.php';
  });
});
```

Display file/folder list

Back-end (PHP)

```
$dir = "uploads/";

$files = scandir($dir);

$fileList = array();

foreach ($files as $file) {
    if ($file != "." && $file != "..") {
        $filepath = $dir . $file;
        $fileInfo = array(
            "name" => $file,
            "type" => is_dir($filepath) ? "Folder" : "File",
            "size" => is_dir($filepath) ? "-" : filesize($filepath),
```

```
            "last_modified" => date("d-m-Y H:i:s", filemtime($filepath))
        );
        array_push($fileList, $fileInfo);
    }
}

header('Content-Type: application/json');
echo json_encode($fileList);
```

Front-end (JS)

```
function loadFiles() {
   fetch('load_files.php')
      .then(response => {
         if (!response.ok) {
            throw new Error('Error loading files');
         }
         return response.json();
      })
      .then(data => {
         const tableBody = document.querySelector('tbody')
         tableBody.innerHTML = '';

         data.forEach(file => {
            const row = document.createElement('tr');
            row.innerHTML = `
            <td>${file.name}</td>
            <td>${file.type}</td>
            <td>${file.size}</td>
            <td>${file.last_modified}</td>
            <td>
               <i class="fa fa-download action"></i>
               <i class="fa fa-edit action"></i>
               <i class="fa fa-trash action"
onclick="deleteFile('${file.name}')"></i>
            </td>
          `;
            tableBody.appendChild(row);
         });
      })
      .catch(error => console.error('Error:', error));
}
```

Create folder

Back-end (PHP)

```php
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $data = json_decode(file_get_contents("php://input"), true);
    $foldername = $data['foldername'];
    $dir = "uploads/";
    mkdir($dir . $foldername);
}
```

Front-end (HTML)

```html
<div class="modal fade" id="new-folder-dialog" tabindex="-1" role="dialog"
aria-labelledby="newFolderModalLabel"
   aria-hidden="true">
   <div class="modal-dialog" role="document">
      <div class="modal-content">
         <div class="modal-header">
            <h5 class="modal-title" id="newFolderModalLabel">New Folder</h5>
            <button type="button" class="close" data-dismiss="modal" aria-
label="Close">
               <span aria-hidden="true">&times;</span>
            </button>
         </div>
         <div class="modal-body">
            <div class="form-group">
               <label for="folderName">Folder Name:</label>
               <input type="text" class="form-control" id="folderName"
placeholder="Enter folder name">
            </div>
         </div>
         <div class="modal-footer">
            <button type="button" class="btn btn-secondary" data-
dismiss="modal">Cancel</button>
            <button type="button" class="btn btn-primary"
id="createFolderBtn">OK</button>
         </div>
      </div>
   </div>
</div>
```

Front-end (JS)

```
function showNewFolderModal() {
   $('#new-folder-dialog').modal('show');
}
```

```
function createFolder(foldername) {
   fetch('create_folder.php', {
      method: 'POST',
      body: JSON.stringify({ foldername: foldername }),
      headers: {
         'Content-Type': 'application/json'
      }
   })
      .then(response => {
         if (!response.ok) {
            throw new Error('Error creating folder');
         }
         loadFiles();
      })
      .catch(error => console.error('Error:', error));
}
```

Create text file

Back-end (PHP)

```
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    $data = json_decode(file_get_contents("php://input"), true);
    $filename = $data['filename'];
    $content = $data['content'];
    $dir = "uploads/";
    file_put_contents($dir . $filename, $content);
}
```

Front-end (HTML)

```
<div class="modal fade" id="new-file-dialog">
   <div class="modal-dialog">
      <div class="modal-content">

         <div class="modal-header">
            <h4 class="modal-title">Tạo tập tin mới</h4>
            <button type="button" class="close" data-
dismiss="modal">&times;</button>
         </div>
```

```
        <div class="modal-body">
           <div class="form-group">
              <label for="name">File Name</label>
              <input type="text" placeholder="File name" class="form-control"
id="file-name" />
           </div>
           <div class="form-group">
              <label for="content">Nội dung</label>
              <textarea rows="10" id="file-content" class="form-control"
placeholder="Nội dung"></textarea>

           </div>
        </div>

        <div class="modal-footer">
           <button type="button" class="btn btn-success" id="createFileBtn" data-
dismiss="modal">Lưu</button>
        </div>
     </div>
  </div>
</div>
```

Front-end (JS)

```
function showNewFileModal() {
   $('#new-file-dialog').modal('show');
}
```

```
function createTextFile(filename, content) {
   fetch('create_text_file.php', {
      method: 'POST',
      body: JSON.stringify({ filename: filename, content: content }),
      headers: {
         'Content-Type': 'application/json'
      }
   })
      .then(response => {
         if (!response.ok) {
            throw new Error('Error creating text file');
         }
         loadFiles();
      })
      .catch(error => console.error('Error:', error));
}
```

Upload file

Back-end (PHP)

```php
$targetDir = "uploads/";

if (!file_exists($targetDir)) {
    mkdir($targetDir, 0777, true);
}

$targetFile = $targetDir . basename($_FILES["fileToUpload"]["name"]);
$uploadOk = 1;
$fileType = strtolower(pathinfo($targetFile, PATHINFO_EXTENSION));

if (file_exists($targetFile)) {
    echo json_encode(array('message' => 'File already exists'));
    exit;
}

if ($_FILES["fileToUpload"]["size"] > 5000000) {
    echo json_encode(array('message' => 'File is too large'));
    exit;
}

if ($filetype != "jpg" && $fileType != "png" && $filetype != "txt") {
    echo json_encode(array('message' => 'File type is invalid'));
    exit;
}

if (!move_uploaded_file($_FILES["fileToUpload"]["tmp_name"], $targetFile)) {
    echo json_encode(array('message' => 'Error uploading file'));
    exit;
}

echo json_encode(array('message' => 'File uploaded successfully'));
```

Front-end (JS)

```javascript
$('#uploadForm').submit(function (e) {
   e.preventDefault();

   var formData = new FormData(this);

   $.ajax({
      url: 'upload_file.php',
      type: 'POST',
      data: formData,
```

```
      success: function (response) {
         alert(response);
         $('#uploadForm')[0].reset();
         loadFiles();
      },
      error: function (error) {
         console.error('Error:', error);
      },
      cache: false,
      contentType: false,
      processData: false
   });
});
```

Form (HTML)

```
<form id="uploadForm" enctype="multipart/form-data">
   <div class="form-group">
      <div class="custom-file">
         <input type="file" class="custom-file-input" id="fileToUpload"
name="fileToUpload">
         <label class="custom-file-label" for="customFile">Choose file</label>
      </div>
   </div>
   <button class="btn btn-success px-5">Upload</button>
</form>
```

References

https://www.w3schools.com/php/default.asp