HTTP

1PHPD

1PHPD – HTTP

Course Objectives

By the end of the course, students should:

- Know how to work with HTTP
- Be able to handle user request
- Build secure forms

Time:

- course: 2h
- exercises: 4h

Summary

1. HTTP
2. Forms
3. Security
4. Sessions and Cookies

1. HTTP

1. HTTP

PHP and HTTP (Hypertext Transfer Protocol) work together closely in web development, with PHP often serving as a server-side scripting language that generates dynamic content in response to HTTP requests.

Understanding how PHP interacts with HTTP can help developers create more efficient, responsive web applications.

1. HTTP

HTTP verbs, also known as methods, define actions that can be performed on resources identified by URLs in the context of the web.

They are a fundamental part of the HTTP protocol, playing a crucial role in the RESTful architecture of web services and APIs.

Each HTTP method has specific semantics, guiding their appropriate use for different operations in web applications.

1. HTTP

The GET method requests a representation of the specified resource. Requests using GET should only retrieve data and have no other effect. It is safe and idempotent, meaning multiple identical requests should have the same effect as a single one.

Use case: Retrieving a webpage or fetching data from an API without causing a change in the data.

1. HTTP

The POST method submits an entity to the specified resource, often causing a change in state or side effects on the server. It is used to create new resources or submit form data.

Use case: Creating a new record in a database (e.g., posting a new blog post), submitting form data.

1. HTTP

The PUT method replaces all current representations of the target resource with the request payload. It is idempotent and used to update existing resources or create a resource at a specific URL if it doesn't exist.

Use case: Updating the entire content of an existing resource, such as editing a blog post.

1. HTTP

The DELETE method deletes the specified resource. It is idempotent, as deleting the same resource multiple times should result in the same effect — the resource is removed.

Use case: Removing a resource from the server, such as deleting a user account.

1. HTTP

The PATCH method applies partial modifications to a resource. Unlike PUT, which replaces the entire resource, PATCH is used for making changes to specific fields of a resource.

Use case: Updating a subset of the properties of a resource, such as changing a user's email address.

1. HTTP

The HEAD method requests the headers that would be returned if the HEAD request's URL was instead requested with an HTTP GET method. It does not return a body in the response; it's used for obtaining metadata.

Use case: Checking what a GET request to the same URL would return, including the status or content-type headers, without actually fetching the body of the resource.

1. HTTP

The OPTIONS method describes the communication options for the target resource. It can be used to check which HTTP methods are supported by a server or specific resource.

Use case: Determining which operations (HTTP methods) are allowed on a server or specific resource, useful for CORS (Cross-Origin Resource Sharing) in web applications.

1. HTTP

Each method serves specific purposes in the context of resource manipulation and should be chosen according to the action being performed to adhere to the standards and expectations of the HTTP protocol.

Understanding HTTP methods is crucial for developing web applications and APIs that follow RESTful principles.

1. HTTP

HTTP status codes are standardized codes in the Hypertext Transfer Protocol (HTTP) that are issued by a server in response to a client's request to the server.

These status codes are part of the HTTP response and indicate whether a specific HTTP request has been successfully completed, and if not, they describe the error.

1. HTTP

1xx: Informational Responses

These indicate that the request was received and understood by the server, and processing is continuing.

- 100 Continue: The initial part of a request has been received, and the client should continue with the rest of the request.
- 101 Switching Protocols: The server is switching protocols as requested by the client (e.g., switching to WebSocket).

1. HTTP

2xx: Successful Responses

These indicate that the request was successfully received, understood, and accepted.

- 200 OK: The request has succeeded. The meaning of success varies depending on the HTTP method used.
- 201 Created: The request has succeeded, and a new resource has been created as a result.
- 204 No Content: The server successfully processed the request, but is not returning any content.

1. HTTP

3xx: Redirection Messages

These indicate that the client must take additional action to complete the request.

- 301 Moved Permanently: The requested URL has been permanently moved to a new location, and the change is permanent.
- 302 Found: The server has found a temporary redirection. This URL should be used again for the next time since the redirection might change.

1. HTTP

4xx: Client Error Responses

These indicate an error that occurred on the client's side.

- 400 Bad Request: The server cannot process the request due to a client error (e.g., malformed request syntax).
- 401 Unauthorized: Authentication is required, and it has failed or has not been provided yet.
- 404 Not Found: The server cannot find the requested resource.

1. HTTP

5xx: Server Error Responses

These indicate that the server failed to fulfill a valid request.

- 500 Internal Server Error: The server encountered an unexpected condition that prevented it from fulfilling the request.
- 504 Gateway Timeout: The server, while acting as a gateway or proxy, did not receive a timely response from an upstream server.

1. HTTP

Each status code provides valuable information about the outcome of the HTTP request, enabling developers and clients to understand the context and nature of the response from the server.

Proper use and handling of HTTP status codes are essential for effective web development and user experience.

Exercises

2. Forms

2. Forms

PHP and HTML forms work together to enable dynamic user interactions on the web. HTML forms collect user input, which PHP can then process server-side.

This interaction allows for a wide range of functionalities, including user registration, login systems, data submission to databases, and more.

2. Forms

An HTML form is defined using the <form> tag. The form attributes action and method are crucial for integrating with PHP:

- action: Specifies where to send the form data when the form is submitted. This is often set to the URL of a PHP file that will process the data.
- method: Defines how to send the data, with GET and POST being the most common methods.

2. Forms

Example

```
<form action="submit.php" method="post">
    Name: <input type="text" name="name"><br>
    Email: <input type="text" name="email"><br>
    <input type="submit">
</form>
```

2. Forms

When a user submits the form, the form data is sent to the PHP file specified in the action attribute, where it can be processed.

- Using $_GET: When the form method is GET, form data is visible in the URL and accessible in PHP using the global $_GET array.
- Using $_POST: For the POST method, form data is not visible in the URL and is accessible in PHP using the global $_POST array.

2. Forms

In the PHP file specified in the form's action attribute, you can access and process the form data.

Reusing our previous example, we are handling a POST method, with two fields in the form

2. Forms

Example

```php
<?php
// Check if the form is submitted
if ($_SERVER["REQUEST_METHOD"] == "POST") {
    // Collect value of input field
    $name = htmlspecialchars($_POST['name']);
    $email = htmlspecialchars($_POST['email']);

    if (empty($name)) {
        echo "Name is empty";
    } else {
        echo "Name: $name<br>";
        echo "Email: $email";
    }
}
?>
```

2. Forms

This script checks if the form has been submitted, sanitizes the input to prevent XSS attacks (htmlspecialchars function), and then processes the data.

We will see the sanitization and security in a next chapter.

2. Forms

In PHP, you can read HTTP headers sent by the client (such as web browsers or other HTTP clients) using a couple of approaches.

These headers can include various types of information, such as the client's browser type, preferred language, referring URL, and more.

Knowing how to read these headers allows you to make decisions based on the client's context, enhance security, tailor content, and more.

2. Forms

PHP stores information about headers, paths, and script locations in the $_SERVER superglobal array.

This array includes entries for many HTTP request headers, where the header names are converted to uppercase, prefixed with HTTP_, and hyphens (-) replaced with underscores (_).

2. Forms

Example: Reading the User-Agent and Referer Headers

```php
$userAgent = $_SERVER['HTTP_USER_AGENT'];
$referer = isset($_SERVER['HTTP_REFERER']) ? $_SERVER['HTTP_REFERER'] : 'No referer';

echo "User Agent: $userAgent<br>";
echo "Referer: $referer";
```

2. Forms

In this example, $_SERVER['HTTP_USER_AGENT'] is used to access the User-Agent header, which provides information about the client's browser.

The Referer header, which indicates the URL of the page that referred the user to the current page, is accessed via $_SERVER['HTTP_REFERER'].

2. Forms

The availability of specific headers depends on the client and the server environment. Not all clients send the same headers, and some headers may be removed or altered by intermediate proxies.

Be cautious when using header information, as it can be manipulated by the client or through other external factors. Validate and sanitize any header values before using them in your application logic, especially if they influence behavior or content.

2. Forms

Reading HTTP headers from the client can provide valuable context for your PHP application, enabling more personalized and secure interactions with users.

Whether you're logging request information, adapting content based on the client's capabilities, or implementing security checks, accessing request headers is a key part of HTTP-based programming in PHP.

2. Forms

In PHP, you can manipulate HTTP headers to control various aspects of your application's behavior, such as setting content types, managing caching, enforcing security policies, and redirecting users.

Manipulating HTTP headers in PHP is done using the header() function.

2. Forms

The header() function is used to send raw HTTP headers to the client.

It must be called before any actual output is sent, either by normal HTML tags, blank lines in a file, or from PHP.

```php
header(string $header, bool $replace = true, int $response_code = 0);
```

2. Forms

Example

- $header: The header string.
- $replace: Whether to replace a previous similar header, or add a second header of the same type. By default, it is true, replacing any previous similar headers.
- $response_code: Forces the HTTP response code to the specified value.

2. Forms

Some specific use case of the header functions

Redirecting to a New URL

```php
header('Location: https://www.example.com');
exit; // Always call exit() after redirection to stop script execution
```

2. Forms

To inform the browser about the type of content being sent.

```php
header('Content-Type: text/plain'); // Plain text
header('Content-Type: application/json'); // JSON format
```

2. Forms

To explicitly set the response status code.

```php
header('HTTP/1.1 404 Not Found');
```

2. Forms

Headers must be sent before any output from your script (this includes whitespace). If you attempt to send a header after any output, PHP will raise a warning, and the header will not be sent.

You can use the header_remove() function to remove previously set headers.

It's good practice to use exit or die functions after sending a location header to ensure the script execution stops.

Exercises

3. Security

3. Security

When dealing with PHP forms and user input, security is paramount.

Malicious users can attempt to exploit vulnerabilities in your application, potentially leading to data breaches, site defacement, or other harmful outcomes.

You should never trust the data sent by users, and always consider it as malicious.

3. Security

Validation ensures that the input meets specific criteria (e.g., an email address should match the format of an email address).

Sanitization cleans the input to ensure it is safe to use.

- Use PHP filter functions like filter_var() with appropriate filter options for validation.
- Sanitize inputs using PHP's filter functions or custom sanitization functions, especially before including them in database queries or outputting them in HTML.

3. Security

Cross-Site Request Forgery (CSRF) attacks trick a user into submitting a form that they did not intend to submit.

Use anti-CSRF tokens in your forms to mitigate this risk.

- Generate a unique token for each user session.
- Include the token in your forms as a hidden field.
- Verify the token on form submission.

3. Security

Data sent via HTTP is not encrypted, making it susceptible to interception or modification.

Use HTTPS to encrypt data transmitted between the client and server, protecting sensitive information such as passwords.

And use POST request to send informations that should not be seen. GET request add the content in the URL, which is inspectable by all machines between you and the server.

3. Security

Escaping output prevents Cross-Site Scripting (XSS) attacks by ensuring that any input displayed on the page is treated as data, not executable code.

- Use htmlspecialchars() or htmlentities() to escape any data output to HTML, especially data that comes from user inputs.

3. Security

Implementing a Content Security Policy can help prevent XSS attacks by defining which sources the browser should allow to load content from.

- Specify your CSP via the Content-Security-Policy HTTP header.

3. Security

If your form allows file uploads, ensure you:

- Limit file types to only those you explicitly allow.
- Scan uploaded files for malware if possible.
- Store uploaded files outside the webroot or ensure they cannot be executed.
- Rename uploaded files to avoid overwriting existing files or executing code.

3. Security

Security in web development is a broad and critical topic.

When working with PHP forms and user input, it’s important to be vigilant and implement security best practices to protect your application and its users

Exercises

4. Sessions and Cookies

4. Sessions and Cookies

PHP sessions and cookies are fundamental for maintaining state and managing user data across different pages of a web application.

While HTTP is a stateless protocol, sessions and cookies provide a way to store user-specific information to create a personalized and interactive user experience.

4. Sessions and Cookies

Cookies are small pieces of data stored on the client's browser. They are sent to and from the server along with HTTP requests and responses.

Cookies can be used for various purposes, such as tracking user preferences, authentication, and maintaining session state.

4. Sessions and Cookies

Setting a Cookie in PHP is really easy, simply use the setcookie() function

```
setcookie(name, value, expire, path, domain, secure, httponly);
```

4. Sessions and Cookies

name: The name of the cookie.
value: The value of the cookie.
expire: The timestamp when the cookie expires.
path, domain: Define the scope of the cookie.
secure: Specifies whether to transmit the cookie over HTTPS only.
httponly: If TRUE, makes the cookie accessible only through the HTTP protocol.

4. Sessions and Cookies

If your application uses cookies (e.g., for sessions), mark them as secure and HTTPOnly.

- Secure: Indicates that the cookie should only be sent over HTTPS.
- HTTPOnly: Prevents access to the cookie via JavaScript, mitigating the risk of client-side script access.

4. Sessions and Cookies

Example: This creates a cookie named "user" with the value "John Doe" that expires in one hour.

```
setcookie("user", "John Doe", time() + 3600, "/");
```

4. Sessions and Cookies

You can then access cookies through the $_COOKIE superglobal array

```php
echo $_COOKIE["user"];
```

4. Sessions and Cookies

A session is a way to store information on the server for individual users.

A session ID is sent to the client as a cookie, and this ID is used by the client to retrieve its session data during subsequent requests.

4. Sessions and Cookies

Starting a Session in PHP is like creating a cookie. It can be done by using the session_start() function

```
session_start();
```

This must be called before any output is sent to the browser.

4. Sessions and Cookies

Accessing and modifying session is done the same way as you would work with a variable.

Store data in the $_SESSION superglobal array

```php
$_SESSION["favcolor"] = "green";
```

4. Sessions and Cookies

Access session data using the same array

```php
echo $_SESSION["favcolor"];
```

4. Sessions and Cookies

Use session_unset() to free all session variables and session_destroy() to destroy the session.

```
session_unset();
session_destroy();
```

4. Sessions and Cookies

Nevertheless there is some key distinctions between sessions and cookies

- Lifetime
  - Cookies are stored on the client-side and have a set expiration time.
  - Sessions are stored on the server, and the session cookie (containing the session ID) typically expires when the browser is closed.

- Storage:
  - Cookies store data directly in the browser (limited to about 4KB per cookie).
  - Session data is stored on the server, which can typically handle much larger data amounts securely.

4. Sessions and Cookies

- Security:
  - Storing sensitive data directly in cookies is not recommended due to potential security risks.
  - Sessions are generally more secure for handling sensitive data, as the data is stored on the server, and only the session ID is exchanged with the browser.

4. Sessions and Cookies

Use Cases for Cookies:

- Remembering user preferences (e.g., theme, language).
- Implementing "Remember Me" functionality for login forms.

4. Sessions and Cookies

Use Cases for Sessions:

- Tracking user login state across pages.
- Storing user-specific data that needs to be accessed across multiple pages (e.g., shopping cart contents).

4. Sessions and Cookies

In summary, both sessions and cookies are essential for creating dynamic, personalized web applications with PHP.

Choosing between them depends on the specific requirements regarding data security, lifetime, and the amount of data to be stored.

Exercises