Trường Đại học Công nghệ Thông tin – Đại học Quốc gia Hồ Chí Minh
Bộ môn An toàn thông tin

PHP Advanced

Ths. Trần Tuấn Dũng – dungtran@uit.edu.vn

Form Handling

- We have this simple form
- We're submitting it to **welcome.php** file on server, how can the **php** file handle user inputs ?

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

Form Handling

- By using superglobal variable **$_POST** (previous lesson)
- Provide the **$_POST** with the name of the parameters

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

- **Welcome.php** will look like this

```
Welcome <?php echo $_POST["name"]; ?><br>
Your email address is: <?php echo $_POST["email"]; ?>
```

- GET and other methods work the same

Form Validation

- Form validation mostly depends on Javascript to validate from client-side, but when comes to saving data, for handling server-side, PHP are capable of validating inputs as well
- Form validation just a technique and depend on raw programming skill
- But PHP mostly handles value came from **URL's query params**
  - It's a best practice to escape char and pre-process the data first before saving or perform any action

PHP Form Validation (w3schools.com)

PHP Forms Required Fields (w3schools.com)

PHP Forms Validate E-mail and URL (w3schools.com)

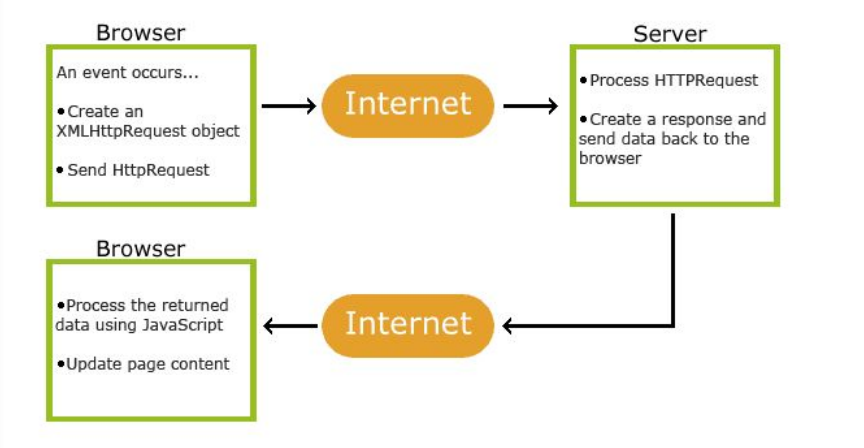
PHP Complete Form Example (w3schools.com)

Ajax

AJAX = **Asynchronous** JavaScript and XML.

AJAX is a technique for creating **fast and dynamic** web pages.

AJAX allows web pages to be updated **asynchronously** by exchanging small amounts of data with the server behind the scenes. This means that it is possible to update parts of a web page, **without reloading** the whole page.



Ajax

Ajax is very handy when comes to handle user small action, such as button click, fill in an input, scroll the page,… And we need to update the web page immediate based on the action

We have this simple form:

```
<p><b>Start typing a name in the input field below:</b></p>
<form action="">
  <label for="fname">First name:</label>
  <input type="text" id="fname" name="fname" onkeyup="showHint(this.value)">
</form>
<p>Suggestions: <span id="txtHint"></span></p>
```

We want to show user a hint base on their input, the hint will be store in database, how can we get those hint from database and show to user ?

Ajax

```
function showHint(str) {
      if (str.length == 0) {
          document.getElementById("txtHint").innerHTML = "";          return;
      }
      else {
          var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function() {
              if (this.readyState == 4 && this.status == 200) {
document.getElementById("txtHint").innerHTML = this.responseText;
            }};
            xmlhttp.open("GET", "gethint.php?q=" + str, true);
      xmlhttp.send();
}}
```

This function will submit a value with param **q** into **gethint.php** file with method **GET**, the php file will then process the input and respond with the result, which is captured in **responseText**

How gethint.php works ?

Ajax

```php
// get the q parameter from URL
$q = $_REQUEST["q"];
$hint = "";
// lookup all hints from array if $q is different from ""
if ($q !== "") {
$q = strtolower($q);
$len=strlen($q);
foreach($a as $name) {
   if (stristr($q, substr($name, 0, $len))) {
      if ($hint === "") { $hint = $name; }
      else { $hint .= ", $name"; }
    } } }
// Output "no suggestion" if no hint was found or output correct values
echo $hint === "" ? "no suggestion" : $hint;
```

Assume `$name` is data from database or any storing resource, we get data in php file, process it, then output it with `echo`, the echo text is the `responeText`

How to connect to database ?

Database

```php
<?php
$servername = "localhost";
$username = "username";
$password = "password";

// Create connection
$conn = new mysqli($servername, $username, $password);

// Check connection
if ($conn->connect_error) {
  die("Connection failed: " . $conn->connect_error);
}
echo "Connected successfully";
?>
```

Using **mysqli** package to connect with MySQL database

Create Database

```php
// Create database
$sql = "CREATE DATABASE myDB";
if ($conn->query($sql) === TRUE) {
  echo "Database created successfully";
} else {
  echo "Error creating database: " . $conn->error;
}
```

Create table

```
$sql = "CREATE TABLE MyGuests (
id INT(6) UNSIGNED AUTO_INCREMENT PRIMARY KEY,
firstname VARCHAR(30) NOT NULL,
lastname VARCHAR(30) NOT NULL,
email VARCHAR(50),
reg_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)";

if ($conn->query($sql) === TRUE) {
  echo "Table MyGuests created successfully";
} else {
  echo "Error creating table: " . $conn->error;
}
```

Insert

```
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";

if ($conn->query($sql) === TRUE) {
  echo "New record created successfully";
} else {
  echo "Error: " . $sql . "<br>" . $conn->error;
}
```

Select

```
$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = $conn->query($sql);

if ($result->num_rows > 0) {
  // output data of each row
  while($row = $result->fetch_assoc()) {
    echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. "
" . $row["lastname"]. "<br>";
  }
} else {
  echo "0 results";
}
```

Best Practice

PHP is server-side language, but can display client-side code, which makes make it vulnerable to common attacks like XSS, Injection

To prevent those attacks, always:

- **Sanitize, pre-process, escape** from user input (using htmlescapechar,…)
- **Avoid using raw query**, if must, **never concatenate** string with out process user input first
- A safer way is to use bind param
- Never directly set HTML content without process it first

In lesson are fundamentals, most modern frameworks already handle these

OOP in PHP - Class declaration syntax

```
class ClassName {
      # fields - data inside each object
      public $name; # public field
      private $name; # private field
      # constructor - initializes each object's state
      public function __construct(parameters) {
            statement(s);
            }
      # method - behavior of each object
      public function name(parameters) {
            statements;
      }
}
```

inside a constructor or method, refer to the current object as **$this**

Class example

```php
<?php
class Point {
	public $x;
	public $y;
	# equivalent of a Java constructor
	public function __construct($x, $y) {
		$this->x = $x;
		$this->y = $y;
	}
	public function distance($p) {
		$dx = $this->x - $p->x;
		$dy = $this->y - $p->y;
		return sqrt($dx * $dx + $dy * $dy);
	}
	# equivalent of Java's toString method
	public function __toString() {
		return "(" . $this->x . ", " . $this->y . ")";
	}
} ?>
```

Constructing and using objects

```
# construct an object
$name = new ClassName(parameters);
# access an object's field (if the field is public)
$name->fieldName
# call an object's method
$name->methodName(parameters);
```

```
$zip = new ZipArchive();
$zip->open("moviefiles.zip");
$zip->extractTo("images/");
$zip->close();
```

the above code unzips a file and test whether a class is installed with class_exists

Object example

```
# create an HTTP request to fetch student.php
$req = new HttpRequest("student.php", HttpRequest::METH_GET);
$params = array("first_name" => $fname, "last_name" => $lname);
$req->addPostFields($params);
# send request and examine result
$req->send();
$http_result_code = $req->getResponseCode(); # 200 means OK
print "$http_result_code\n";
print $req->getResponseBody();
```

PHP's HttpRequest object can fetch a document from the web

Basic inheritance

```
class ClassName extends AnotherClassName {
...
}
```

```
class Point3D extends Point {
      public $z;
      public function __construct($x, $y, $z) {
            parent::__construct($x, $y);
            $this->z = $z;
      }
      ...
}
```

The given class will inherit all data and behavior from AnotherClassName

Static methods, fields, and constants

```
static $name = value; # declaring a static field
const $name = value; # declaring a static constant
```

```
# declaring a static method
public static function name(parameters) {
     statements;
}
```

```
ClassName::methodName(parameters); # calling a static method (outside
class)
self::methodName(parameters); # calling a static method (within class)
```

static fields/methods are shared throughout a class rather than replicated in every object

Abstract classes and interfaces

```
interface InterfaceName {
public function name(parameters);
public function name(parameters);
...
}
class ClassName implements InterfaceName { ...
```

```
abstract class ClassName {
     abstract public function name(parameters);
     ...
}
```

Abstract classes and interfaces

- **interfaces** are supertypes that specify method headers without implementations
  - cannot be instantiated; cannot contain function bodies or fields
  - enables polymorphism between subtypes without sharing implementation code
- **abstract classes** are like interfaces, but you can specify fields, constructors, methods
  - also cannot be instantiated; enables polymorphism with sharing of implementation code

Traits

- **Traits** are same with **Interface** or **Abstract Class**
- However, Traits allow us to actually implement the methods instead of just declare function's names
- Better scenario to use **Traits** over **Abstract Classes** and **Interfaces:**
  - Use **Traits** over **Interfaces** when having multiple class with same methods and same behaviour, using **interface** and reimplement those methods will be time consuming
  - Use **Traits** over **Abstract Classes** when having multiple class with different semantics but have the same function, while extend that Class will:
    - Reduce the flexibility: We can't extend the class we actually need
    - Redundancy: We don't use everything of the class (property, other methods)

Traits

Traits declaration

```
trait message1 {
   public function msg1() {
            echo "OOP is fun! ";
   }
}
```

Traits

Use Traits

```
class Welcome {
  use message1;
}

$obj = new Welcome();
$obj->msg1();
```

Namespaces

- Namespaces are qualifiers that solve two different problems:
  - They allow for better organization by grouping classes that work together to perform a task
  - They allow the same name to be used for more than one class

```php
namespace Html;
class Table {
  public $title = "";
  public $numRows = 0;
  public function message() {
    echo "<p>Table '{$this->title}' has {$this->numRows} rows.</p>";
  }
}
$table = new Table();
$table->title = "My table";
$table->numRows = 5;
```

Namespaces

- Use Classes from HTML Namespace

```
$table = new Html\Table();
$row = new Html\Row();
```

- Use with alias

```
use Html as H;
$table = new H\Table();
```

Home work

- Create a simple database with email table have 2 column (id, email)
- Using php and ajax, validate the email base on user typing action (validate.php)
- Save to database (database.php)