Bài giảng

PHÁT TRIỂN ỨNG DỤNG WEB

Lê Đình Thanh
Khoa Công nghệ Thông tin
Trường Đại học Công nghệ, ĐHQGHN
E-mail: thanhld@vnu.edu.vn Mobile: 0987.257.504

Chương 6

Công nghệ web động

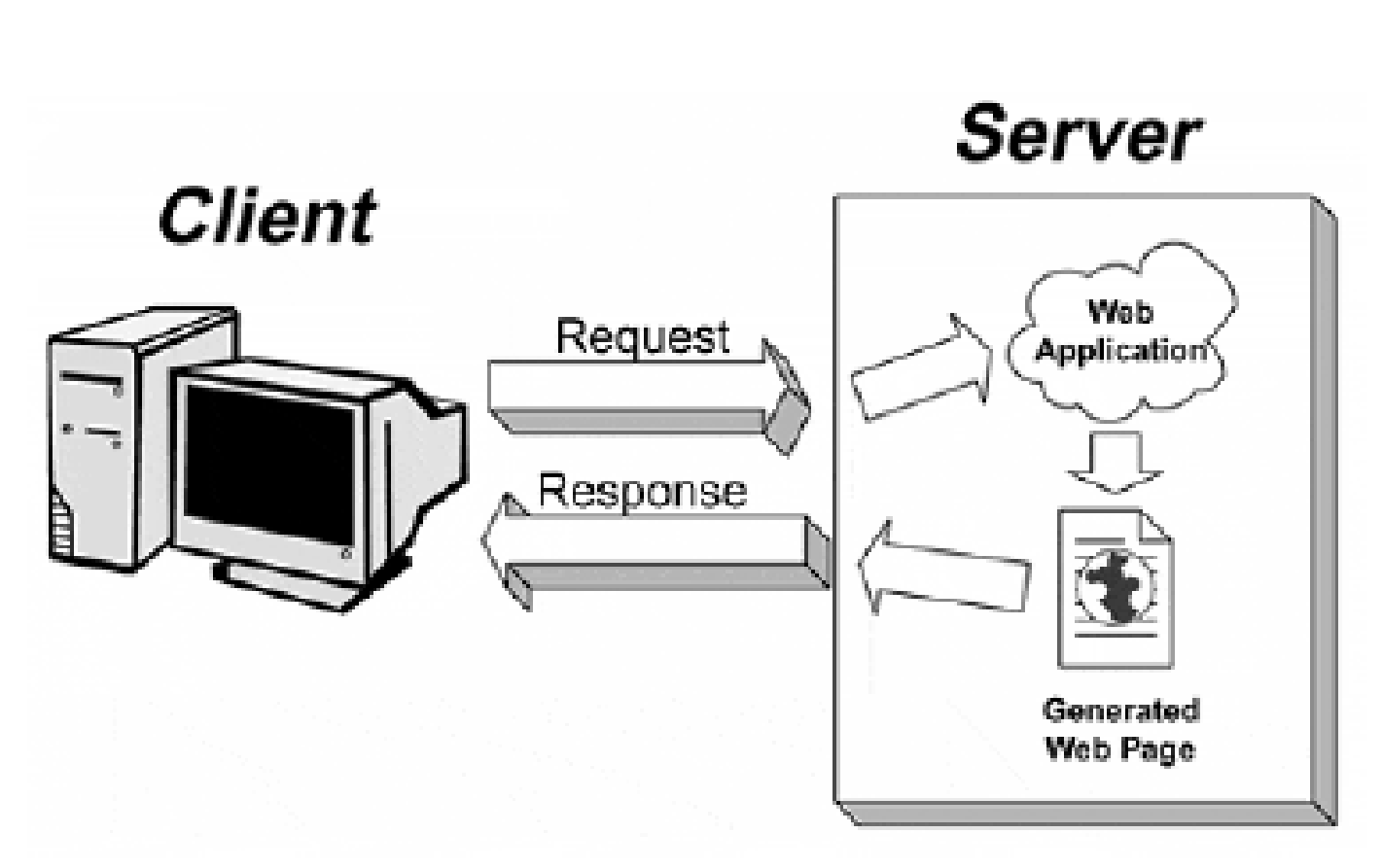
Nội dung

- Kiến trúc của ứng dụng Web động
- Nhiệm vụ bên phục vụ
- PHP
- Tạo web động với PHP
- Mẫu thiết kế MVC
- Giao diện cấu phần hoặc JSON

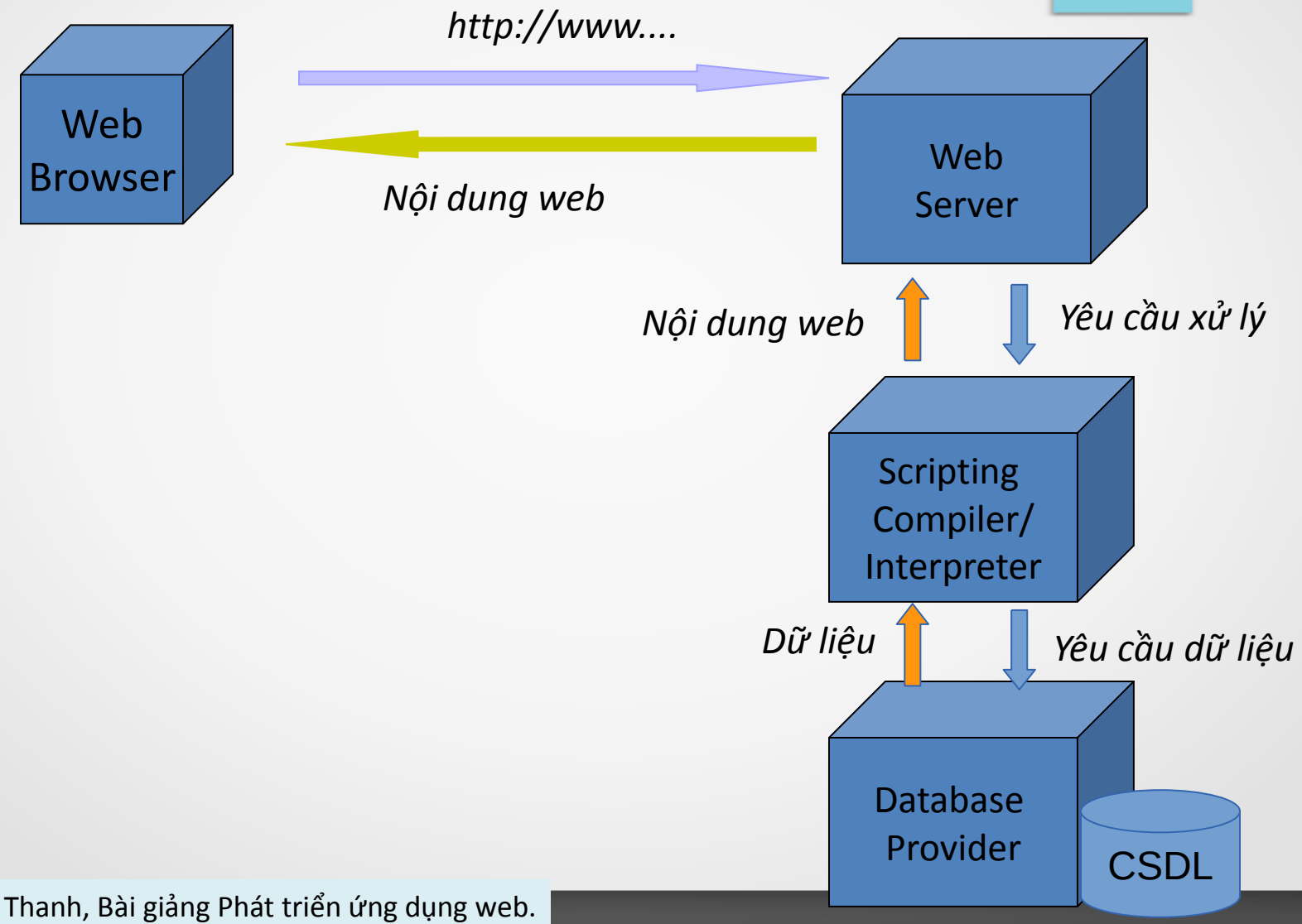
Web động

- Nội dung trang web (HTML + CSS + JavaScript) được ứng dụng web sinh ra khi có yêu cầu từ trình khách.
- Rất phổ dụng: Hầu hết các trang web thương mại đều là web động.
- Sử dụng ngôn ngữ lập trình đa năng để sinh ra nội dung web.
- Sử dụng CSDL.

Kiến trúc web động

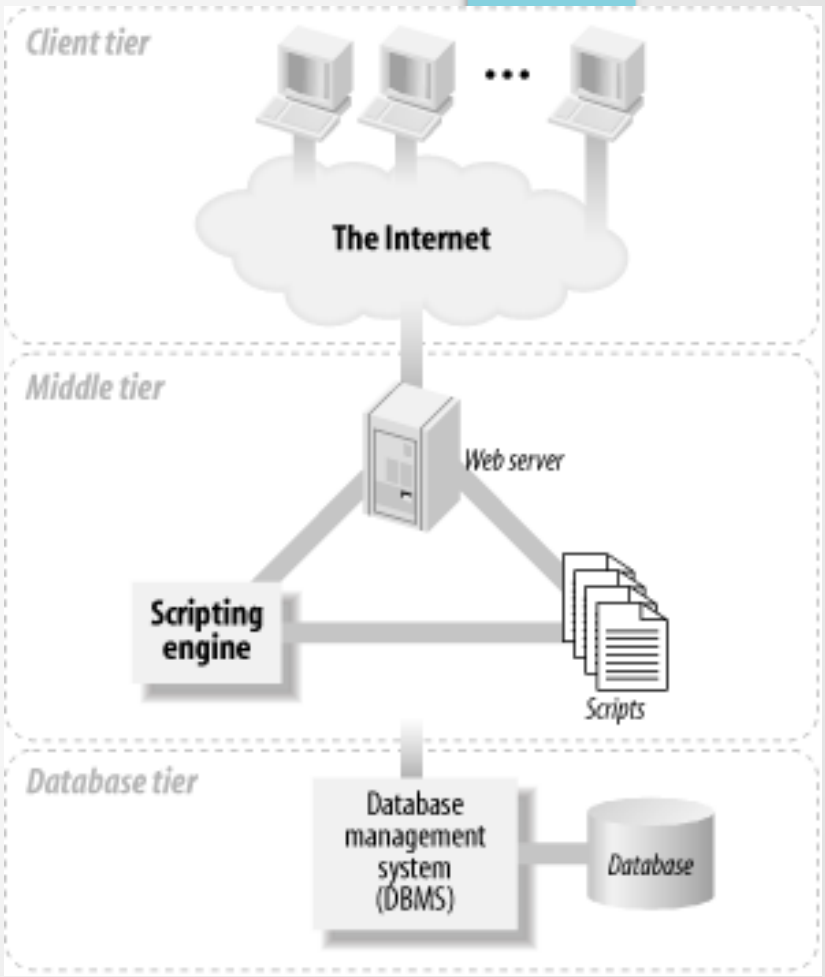


Web động với CSDL



Mô hình ba tầng

- Tầng khách: trình diễn và tương tác với người dùng
- Tầng giữa: thực hiện các logic của ứng dụng
- Tầng CSDL: bao gồm hệ quản trị CSDL, CSDL của ứng dụng



Apache Web Server + PHP Interpreter

- Nhận và phân tích yêu cầu từ client
  - Các tham số được lưu trong các mảng: $_SERVER, $_GET, $_POST, $_FILES, …
- Tạo trả lời chứa nội dung web và gửi cho web client
  - Hàm http_response_code đặt mã trạng thái cho gói HTTP Response
  - Hàm header thay đổi giá trị các trường tiêu đề gói HTTP Response
  - Hàm echo ghi nội dung vào thân gói HTTP Response
- Lưu trạng thái phiên làm việc
  - $_COOKIE, $_SESSION
- Lưu dữ liệu bền vững
  - Làm việc với các hệ quản trị CSDL
- Đảm bảo an ninh
  - Xác thực, điều khiển truy cập, kiểm tra hợp thức dữ liệu vào, làm

Laravel Framework

- Nhận và phân tích yêu cầu từ client
  - Các tham số được lưu trong đối tượng Request

```
public function store(Request $request) {
    $title = $request->input('title');
    $author = $request->input('authors');
}
```

- Tạo trả lời chứa nội dung web và gửi cho web client
  - Sử dụng đối tượng Response

```
public function index() {
    $content = ['Gió Thu', 'Sóng Sánh', 'Chiều Hồng'];
    return response($content)
            ->withHeaders([
              'Content-Type' => 'application/json',
              'Set-Cookie' => 'view=list;HttpOnly'
            ]);
}
```

- Lưu trạng thái phiên làm việc

```
$value = $request->session()->get('key');
$value = session('key');
$value = $request->cookie('name');
$cookie = cookie('name', 'value', $minutes);
response('Hello World')->cookie($cookie);
```

- Lưu dữ liệu bền vững
  - Làm việc với các hệ quản trị CSDL
- Đảm bảo an ninh
  - Xác thực, điều khiển truy cập, kiểm tra hợp thức dữ liệu vào, làm sạch dữ liệu ra, ...

PHP - Hypertext Preprocessor

21/11/2022

Server-side Programming Languages

Most popular server-side programming languages

© W3Techs.com	usage	change since 1 October 2022
1. PHP	77.4%	
2. ASP.NET	7.5%	-0.1%
3. Ruby	5.6%	
4. Java	4.5%	
5. Scala	2.8%	+0.1%

percentages of sites

Fastest growing server-side programming languages since 1 October 2022

© W3Techs.com	sites
1. PHP	15.6
2. Scala	12.3
3. static files	10.6

daily increase of number of sites per million

Find more details in the server-side language surveys

PHP – Đặc điểm

- Tựa Java và C, trừ các điểm sau:
  - Định kiểu không tường minh
  - Tên biến bắt đầu bằng $
  - Mảng là ánh xạ
  - Định nghĩa hàm bằng từ khóa *function*
  - Thư viện hàm cho thực hiện các nhiệm vụ của mặt sau ứng dụng web

Trang PHP

- Các trang có tên mở rộng .php
- Mã PHP được để trong cặp thẻ <?php và ?> - được gọi là các phân đoạn PHP. Có thể nhúng các phân đoạn PHP vào bất kỳ vị trí nào trong trang. Bên ngoài các phân đoạn PHP có thể chứa mã HTML, CSS, javascript.
- Phần mã PHP được thực thi để sinh ra phần động của trang web.
- Sử dụng hàm echo để đưa nội dung (HTML, CSS, javascript) vào thân gói HTTP Response.
- Sử dụng hàm header để thay đổi giá trị các trường tiêu đề gói HTTP Response

Trang php

Mã HTML, CSS, javascript

```
<?php
//Mã php được thực thi bên server để sinh ra phần động của trang web
?>
```

Mã HTML, CSS, javascript

```
<?php
/*Mã php được thực thi bên server để sinh ra phần động của trang web */
?>
```

Mã HTML, CSS, javascript

Ví dụ trang php

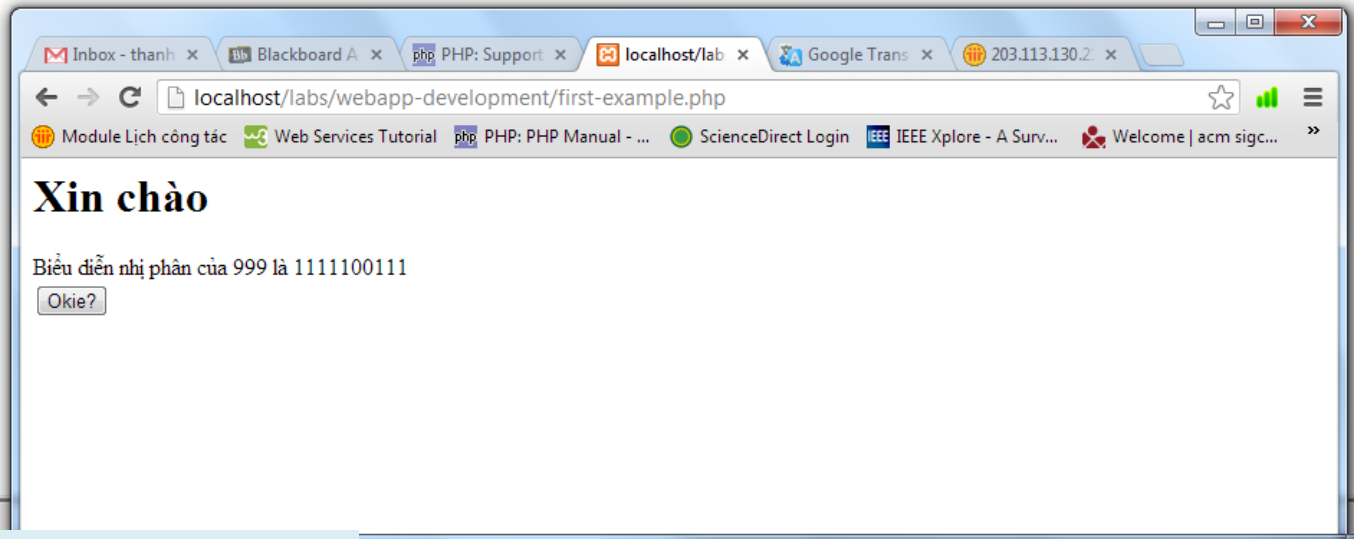
E:\thanh\htdocs\labs\webapp-development\first-example.php - Notepad++

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

index.php | ns.php | ns1.php | ns3.php | first-example.php

```
1  <html>
2      <head>
3      <meta http-equiv="content-type" content="text/html; charset=utf-8">
4      </head>
5      <body>
6          <h1>Xin chào</h1>
7          <?php
8              echo "Biểu diễn nhị phân của 999 là ";
9              echo decbin(999);
10             echo "<br><input type='button' value='Okie?'>";
11         ?>
12     </body>
13 </html>
14
```

PHP Hypertext Preprocessor file



Khi nào thì cần có mã HTML, CSS, javascript trong trang php?

- Những trang chỉ bao gồm mã xử lý nghiệp vụ thì không cần mã HTML, CSS, javascript.
- Những trang tạo giao diện
  - có thể chứa mã HTML, CSS, javascript
  - hoặc dùng hàm echo của php để sinh ra mã HTML, CSS, javascript.

Kiểu, biến và hằng

- Các kiểu nguyên thủy: boolean, float, integer, và string
- Các kiểu phức hợp: array và object
- Tên biến được bắt đầu bằng $
- Định kiểu không rõ ràng
- Định nghĩa hằng: define(“ten_hang", gia_tri);

Phạm vi của biến

- Phạm vi truy cập của một biến là ngữ cảnh nó được định nghĩa
  - Một biến cục bộ được định nghĩa trong một hàm chỉ có phạm vi trong hàm
  - Một biến được định nghĩa toàn cục (không trong hàm nào) có phạm vi trong tệp định nghĩa nó cùng các tệp được bao hàm
- Ví dụ
  - …

Phạm vi của biến (ví dụ)

```
<?php
$a = 10; //biến toàn cục
  function test() {
      $b = 15; //biến cục bộ
      echo $a; //lỗi Undefined variable
      global $a;
      echo ($a+$b); //25
  }
  echo $a; //10
  echo $b; //lỗi Undefined variable
  test(); //25
```

Biến tĩnh

- Biến tĩnh chỉ có phạm vi truy cập cục bộ trong hàm, nhưng giá trị của nó không bị mất khi thực thi của chương trình thoát khỏi hàm

```
<?php
  function tick() {
      static $count = 0;
      echo $count;
      $count++;
  }
  tick(); //0
  tick(); //1
  tick(); //2
```

Biến biến

- Biến biến sử dụng giá trị của một biến khác làm tên
- Ví dụ

```
$a = “delta”;
$r = “array”;
$$a = 2.34; //tương đương $delta = 2.34
$b = array(‘alpha', ‘betha', ’delta', ‘gama');
${$b[2]} == 2.34;
${$r}[1] == ‘betha’;
$obj->$a tương đương $obj->delta
```

Kiểm tra sự tồn tại của biến

- isset($v) - $v đã được thiết lập hay chưa
- empty($v) - $v có giá trị null hay không

- unset($v) – hủy $v

Kiểm tra kiểu của biến

- is_int($v) - $v là số nguyên?
- is_float($v) - $v là số thực?
- is_bool($v) - $v là biến bool?
- is_string($v) - $v là nột xâu?
- is_array($v) - $v là một mảng?
- is_object($v) - $v là một đối tượng?
- is_numeric($v) - $v là một số?
- is_null($v) - $v = NULL?

Chuyển kiểu

- strval($v) - chuyển giá trị của $v thành một xâu
- intval($v) - chuyển giá trị của $v thành một số nguyên
- floatval($v) - chuyển giá trị của $v thành một số thực

Gỡ lỗi với kiểu và giá trị

- In ra kiểu, giá trị và biểu diễn của biểu thức
  - print_r(bieu_thuc)
  - var_dump(bieu_thuc1, bieu_thuc2, …)

Xâu

- Xâu được đánh dấu bởi dấu nháy đơn hoặc nháy kép
  - Ví dụ, “Đây là một xâu ký tự”, ‘Đây là một xâu khác’
- Xâu sử dụng nháy kép có thể chứa biến bên trong và chứa các dãy ký tự thoát

```
$number = 45;
$vehicle = "bus";
$message = "This $vehicle holds $number people. \n";
```

- Nối xâu bằng dấu chấm (.)

```
$message = “This ”. $vehicle .” holds ”. $number .” people.
\n”;
```

- Các dãy ký tự thoát
  - \\, \’, \”, \$, \n, …

Xâu (tiếp)

- strlen($s) – độ dài xâu $s
- strtolower($s) - xâu viết thường của $s
- strtoupper($s) - xâu viết hoa của $s
- ucfirst($s) – xâu viết hoa ký tự đầu của $s
- ucwords($s) – xâu viết hoa ký tự đầu các từ $s
- ltrim($s) – xâu được bỏ dấu trắng đầu xâu của $s
- rtrim($s) – xâu được bỏ dấu trắng cuối xâu của $s
- trim($s) – xâu được bỏ dấu trắng đầu và cuối xâu của $s
- Tạo dữ liệu định dạng:
- string sprintf (string format [, mixed args...])
- printf (string format [, mixed args...])

So sánh xâu

- strcmp($str1, $str2)
- strncmp($str1, $str2, $length)

- 0 – nếu hai xâu bằng nhau
- -1 – nếu $str1 < $str2
- 1 – nếu $str1 > $str2

Tìm và thay thế xâu con

- substr($s, $start [, $length]) – lấy xâu con của $s, bao gồm các ký tự bắt đầu từ chỉ mục $start và có $length ký tự (hoặc đến hết nếu vắng $length)
- strpos($s, $f [, $offset]) – trả về chỉ mục của xuất hiện đầu tiên của $f trong $s, [bắt đầu tìm từ $offset]
- strstr($s, $f) - tìm $f trong $s và trả về xâu con bắt đầu từ điểm xuất hiện đầu tiên của $f đến hết $s
- stristr(string haystack, string needle) – tương tự strstr() nhưng không phân biệt hoa thường
- explode($sep, $s [, $limit]) – trả về mảng là kết quả của tách $s bằng xâu phân cách $sep
- implode($glue, $array) – trả về xâu là kết quả nối các phần từ mảng $array, sử dụng $glue để nối

Tìm và thay thế xâu con

- substr_replace($s, $r, $start [, $length]): Trả về xâu là $s được thay [$length] ký tự bắt đầu từ chỉ mục $start bằng [$length] ký tự đầu của $r
- Ví dụ
  - substr_replace(“chào cháu”, “chú”, 5); cho kết quả “chào chú”

Thay thế xâu con

- strtr($s, $from, $to) – Trả về xâu là kết quả của thay thế các ký tự của $s xuất hiện trong $from bằng ký tự cùng chỉ mục trong $to, Ví dụ

```
$mischief = strtr("command.com", "aeiou", "äëïöü");
print $mischief; // prints cömmänd.cöm
```

- strtr($s, $map) – Trả về xâu là kết quả của thay thế các xâu con của $s xuất hiện trong mảng $map. Ví dụ

```
$glossary = array("BTW"=>"by the way", "IMHO"=>"in my humble opinion", "IOW"=>"in other words", "OTOH"=>"on the other hand");
print strtr($geekMail, $glossary);
```

- Tham khảo: *http://php.net/ref.strings*

Mảng

- Mảng trong PHP là ánh xạ có thứ tự chứa các phần tử <*key*, *value*>
  - *key* chỉ nhận giá trị kiểu nguyên, xâu
  - *value* nhận giá trị kiểu bất kỳ
  - Các phần tử có thể sử dụng *key* tăng tự động

Ví dụ:

```
$arr = array("a" => "Hoàng Hóa", "bc" => "Trần Sang", "Nguyễn Minh");
```

tương đương

```
$arr = array("a" => "Hoàng H", "bc" => "Trần Sang", 0 => "Nguyễn Minh");
```

Mảng (tiếp)

- Truy cập các phần tử mảng
  - array[key]
- Ví dụ

```
echo $arr[“bc”]; //Trần Sang
$tmp = $arr[0];  //Nguyễn Minh
```

Mảng (tiếp)

- Thêm/sửa đổi giá trị phần tử mảng
  - $arr[key] = value; //sửa đổi nếu đã tồn tại phần tử có khóa là key, thêm phần tử mới nếu ngược lại
  - $arr[] = value; //thêm phần tử mới với khóa tăng tự động
- Ví dụ:

```
$arr[] = "Lê Vân";     //Thêm mới
$arr[“a”] = “Hoàng Văn Hóa”;       //Sửa đổi
```

Mảng (tiếp)

- Xóa phần tử mảng
  - unset(array[key]);
- Ví dụ

```
unset($arr[“bc”]);
```

Mảng (tiếp)

- Duyệt các phần tử mảng
  - foreach(array as key=>value)
  - foreach(array as value)
- Ví dụ

```
$a= array(‘mot‘=>1, ‘hai‘=>2, ‘ba‘=> 3);
foreach( $a as $gia_tri)
        echo $gia_tri." "; //1 2 3
foreach($a as $khoa=>$gia_tri)
        echo $khoa. ,:‘ .$gia_tri." "; //mot:1 hai:2 ba:3
```

Mảng "nhiều chiều"

```
$poems = array(
        array("name" => "Nguyễn A",
                "titles" => array("Gió thu", "Sóng sánh", "Chiều hồng")),
        array("name" => "Trần B",
                "titles" => array("Ra trận", "Hồng quân")),
        array("name" => "Trịnh C", "titles" => array("Sông quê"))
);
foreach ($poems as $p) {
        echo $p["name"]." có ". count($p["titles"]). " tác phẩm là:";
        foreach ($p["titles"] as $t) echo " ".$t;
        echo ". ";
}
```

//Nguyễn A có 3 tác phẩm là: Gió thu Sóng sánh Chiều hồng. Trần B //có 2 tác phẩm là: Ra trận Hồng quân. Trịnh C có 1 tác phẩm là: Sông quê.

Mảng (tiếp)

- Đếm số phần tử mảng
  - count(array);
- Sắp xếp các phần tử mảng
  - sort(array);
- Gán mảng
  - array1 = array2;
- ...
- Tham khảo: *http://php.net/ref.array*

Lớp và đối tượng

```
class SimpleClass {
    //định nghĩa hằng
    const constant = 'constant value';
    // định nghĩa biến/thuộc tính
    public $var = 'a default value';
    // định nghĩa hàm/phương thức
    public function displayVar() {
        echo $this->var;
    }
}
$obj = new SimpleClass();
$obj->displayVar();
echo $obj->var;
echo SimpleClass::constant;
```

- $this được dùng để chỉ đối tượng gọi
- Tính khả kiến của các thuộc tính và phương thức có thể là *private, protected, public*

Thuộc tính/phương thức tĩnh

```
class A {
    public static $foo = 'I am foo';
    public $bar = 'I am bar';

    public static function getFoo() { echo self::$foo; }
    public static function setFoo() { self::$foo = 'I am a new foo'; }
    public function getBar() { echo $this->bar;  }
}
$ob = new A();
A::getFoo();     // output: I am foo
$ob->getFoo();    // output: I am foo
A::getBar();     // output:  fatal error: using $this not in object context
$ob->getBar();    // output: I am bar
```

Kế thừa

- Một lớp chỉ có thể kế thừa từ một lớp khác
- Lớp con có thể ghi đè/che phương thức lớp cha
  - Để tuy cập phương thức được kế thừa bị che, sử dụng *parent::*
  - Để định nghĩa một phương thức không thể che, thêm từ khóa *final* vào định nghĩa phương thức

```
class ExtendClass extends SimpleClass {
   // Redefine the parent method
   function displayVar()  {
      echo "Extending class\n";   parent::displayVar();
   }
}
$extended = new ExtendClass();
$extended->displayVar();
```

Hàm tạo

- void **__construct** ([$args [, $...]])

```
class BaseClass {
   function __construct() {
       print "In BaseClass constructor\n";
   }
}
class SubClass extends BaseClass {
   function __construct() {
       parent::__construct();
       print "In SubClass constructor\n";
   }
}
$obj = new BaseClass();
$obj = new SubClass();
```

Hàm hủy

- void **__destruct** (void)

```
class MyDestructableClass {
  function __construct() {
     print "In constructor\n";
     $this->name = "MyDestructableClass";
  }

  function __destruct() {
     print "Destroying " . $this->name . "\n";
  }
}

$obj = new MyDestructableClass();
```

Toán tử chỉ phạm vi (::)

- :: được sử dụng để
  - truy cập hằng
  - truy cập thuộc tính, phương thức tĩnh
  - truy cập phương thức của lớp cha bị che

Lớp ảo, phương thức ảo

- Phương thức ảo
  - Được định nghĩa với từ khóa abstract
  - Chỉ có chữ ký, không có thân
- Lớp ảo
  - Được định nghĩa với từ khóa abstract
  - Không có thể hiện
  - Lớp có phương thức ảo phải được được nghĩa là lớp ảo

Lớp ảo, phương thức ảo

```
abstract class AbstractClass {
    // Force Extending class to define this method
    abstract protected function getValue();
    abstract protected function prefixValue($prefix);

    // Common method
    public function printOut() {    print $this->getValue() . "\n";    }
}

class ConcreteClass extends AbstractClass {
    protected function getValue() {     return "ConcreteClass";     }
    public function prefixValue($prefix) {  return
     "{$prefix}ConcreteClass“;    }
}
```

Giao diện

- Giao diện xác định các phương thức mà lớp phải cài đặt
  - Tất cả các phương thức phải public
- Lớp cài đặt phải cài đặt tất cả các phương thức thuộc giao diện

```
interface iTemplate {
   public function setVariable($name, $var);
   public function getHtml($template);
}
// Implement the interface
class Template implements iTemplate  {
   private $vars = array();
   public function setVariable($name, $var)    {     $this->vars[$name] = $var;     }
   public function getHtml($template)    {
      foreach($this->vars as $name => $value) {
         $template = str_replace('{' . $name . '}', $value, $template);
      }
      return $template;
   }
}
```

Giao diện

- Giao diện có thể kế thừa từ **nhiều** giao diện khác
- Một lớp có thể cài đặt nhiều giao diện

```
interface a {
    public function foo();
}

interface b {
    public function bar();
}
```

Sao chép đối tượng

- Các biến trong PHP chỉ lưu tham chiếu đến đối tượng
- Nếu cần sao chép đối tượng (thành một thể hiện khác), sử dụng hàm

```
$copy_of_object = clone $object;
```

- Hàm clone sao chép tất cả các thuộc tính => các thuộc tính tham chiếu vẫn giữ nguyên giá trị tham chiếu
- Sau khi hoàn thành hàm clone, nếu hàm void **__clone**(void) được định nghĩa, hàm __clone() của đối tượng mới được tạo được gọi cho phép những thay đổi cần thiết giá trị các thuộc tính

Ví dụ sao chép đối tượng

```
class Class1 {
	private $var_ = 1;
	public function printMe() {
		echo " var_ = $this->var_";
	}
	public function changeValue($v_) {
		$this->var_ = $v_;
	}
}

$obj1 = new Class2();
$obj2 = clone $obj1;
$obj1->printMe();
$obj2->printMe();
$obj2->changeValues(5, 6);
$obj1->printMe();
$obj2->printMe();
```

```
var = 2 var_ = 1
var = 2 var_ = 1
var = 2 var_ = 6
var = 5 var_ = 6
```

```
class Class2 {
	private $var = 2;
	private $obj;

	public function __construct() {
		$this->obj = new Class1();
	}

	public function printMe() {
		echo "<br>var = $this->var";
		$this->obj->printMe();
	}

	public function changeValues($v, $v_) {
		$this->var = $v;
		$this->obj->changeValue($v_);
	}
```

Ví dụ sao chép đối tượng

```
class Class1 {
	private $var_ = 1;
	public function printMe() {
		echo " var_ = $this->var_";
	}
	public function changeValue($v_) {
		$this->var_ = $v_;
	}
}
```

```
$obj1 = new Class2();
$obj2 = clone $obj1;
$obj1->printMe();
$obj2->printMe();
$obj2->changeValues(5, 6);
$obj1->printMe();
$obj2->printMe();
```

```
var = 2 var_ = 1
var = 2 var_ = 1
var = 2 var_ = 1
var = 5 var_ = 6
```

```
class Class2 {
	private $var = 2;
	private $obj;

	public function __construct() {
		$this->obj = new Class1();
	}

	public function __clone() {
		$this->obj = new Class1();
	}
	public function printMe() {
		echo "<br>var = $this->var";
		$this->obj->printMe();
	}

	public function changeValues($v, $v_) {
		$this->var = $v;
		$this->obj->changeValue($v_);
	}
```

Không gian tên

- Không gian tên (namespace) được sử dụng để nhóm các lớp, giao diện, hàm và hằng nhằm tránh đụng độ khi sử dụng lại mã do trùng tên
  - Ở các ngôn ngữ khác:
    - .NET: namespace
    - Java: package

Định nghĩa không gian tên

- Sử dụng từ khóa namespace để định nghĩa không gian tên

```
<?php
namespace MyNameSpace {
    const CONNECT_OK = 1;
        interface Conn { /*…*/}
    class Connection { /* ... */ }
        function connect() { /* ... */  }
}
?>
```

Không gian tên lồng nhau

- Sử dụng cú pháp biểu diễn thư mục

```
<?php
namespace Parent\Child\GrandChild;
    const CONNECT_OK = 1;
        interface Conn { /*…*/}
    class Connection { /* ... */ }
        function connect() { /* ... */  }

?>
```

Không gian tên toàn cục

- Các lớp, giao diện, hàm, hằng không được định nghĩa trong một không gian tên nào được coi nằm trong không gian tên toàn cục (\)
- Có thể sử dụng namespace không có tên để biểu thị không gian tên toàn cục

  namespace { /*Không gian tên toàn cục*/ }
- Các không gian tên khác được xem như nằm trong không gian tên toàn cục.

\
\namespace1
\namespace1\subnamespace1
\namespace2
\namespace2\subnamespace2
\namespace2\subnamespace2\subsubnamespace2

Tên đầy đủ trong không gian tên

- Tên đầy đủ của lớp, giao diện, hàm, hằng bao gồm không gian tên phía trước

\namespace\ClassName
\namespace\InterfaceName
\namespace\functionName
\namespace\CONSTANT_NAME

Phân giải tên

- Khi lớp, giao diện không được viết với tên đầy đủ
  - Chúng được hiểu là thuộc không gian tên hiện tại
- Khi hàm, hằng không được viết với tên đầy đủ
  - Chúng được hiểu là thuộc không gian tên hiện tại
  - Hoặc thuộc không gian tên toàn cục nếu không tìm thấy trong không gian tên hiện tại

Ví dụ phân giải tên

```
namespace ns1 {
    class A {
        private $var = 1;
        public function a() {
            echo "<br>ns1\A->var:
    $this->var";
        }
    }
}
namespace ns2\ns3 {
    class A {
        private $var = 3;
        public function a() {
            echo "<br>ns3\A->var:
    $this->var";
        }
    }
}
```

```
namespace ns2 {
    class A {
        private $var = 2;
        public function a() {
            echo "<br>ns2\A->var:
    $this->var";
        }


    $obj1 = new \ns1\A();
    $obj2 = new A();
    $obj3 = new ns3\A();

    $obj1->a();
    $obj2->a();
    $obj3->a();
}
```

```
ns1\A->var: 1
ns2\A->var: 2
ns3\A->var: 3
```

Nhập và đặt bí danh

- Để không phải viết tên đầy đủ (dài), PHP cho phép nhập và đặt bí danh cho không gian tên, lớp, và giao diện
- use ns\subns\Classname; //sau đó chỉ cần sử dụng Classname thay cho tên đầy đủ
- use ns\subns\Classname as Another; //sau đó sử dụng Another thay cho tên đầy đủ
- use ns\subns\NSname; //sau đó sử dụng NSname thay cho tên đầy đủ

Xử lý ngoại lệ

- Ném ngoại lệ

throw new Exception("Mô tả ngoại lệ");

- Bắt ngoại lệ

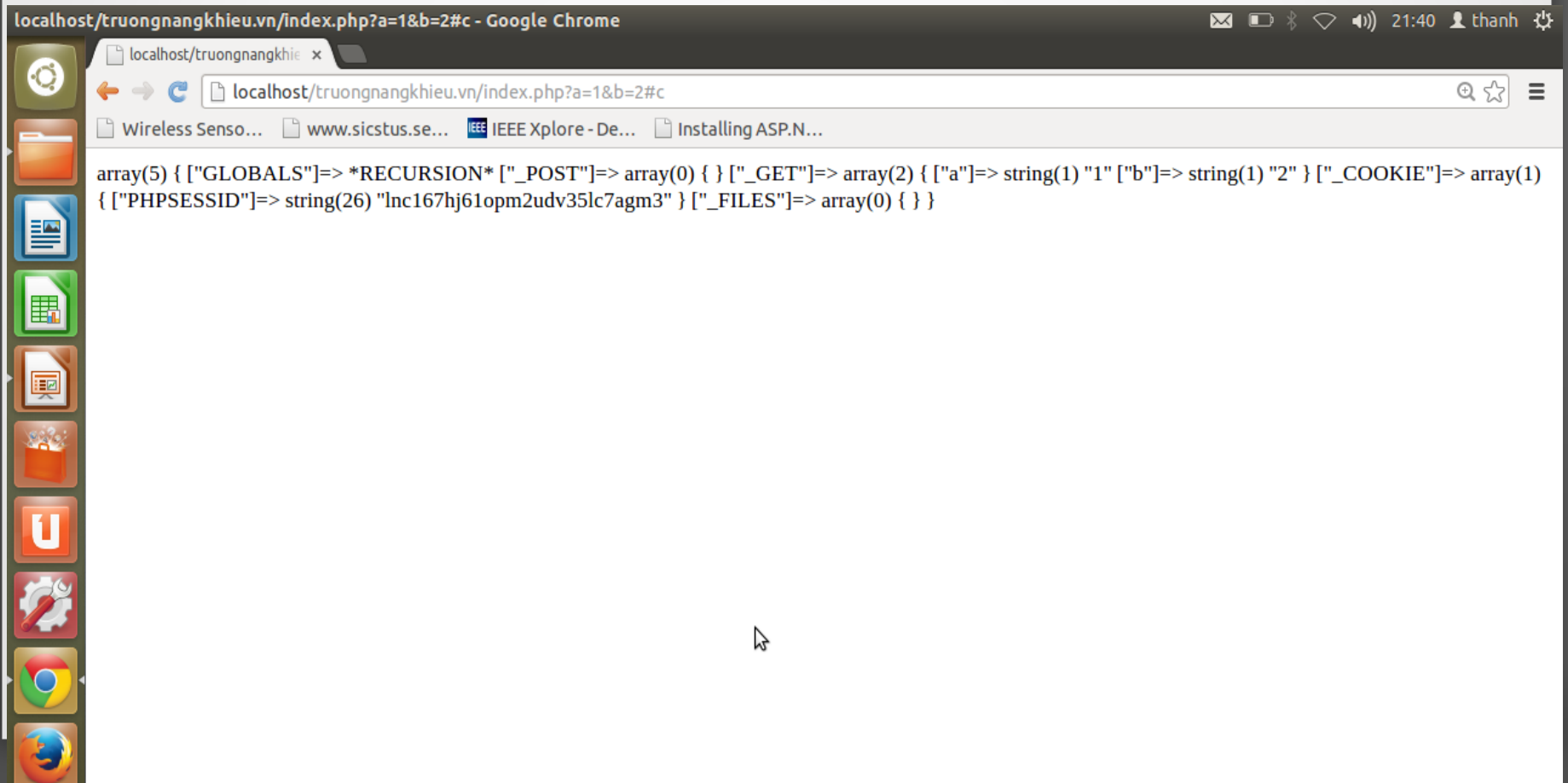
try {
//mã xử lý nghiệp vụ
} catch (Exception $e) {
//nếu có ngoại lệ xảy ra ở khối try thì mã xử lý ngoại lệ ở khối catch được thực hiện. Sử dụng $e->getMessage() để lấy mô tả ngoại lệ
} [catch (OtherException $oe) {
//Có thể nhiều khối catch sau khối try. Mỗi khối catch bắt một loại ngoại lệ
}]*
[finally {
Mã được chạy bất kể ngoại lệ đã xảy ra hay không
}]

Các biến dựng sẵn

- $GLOBALS — Mảng các biến toàn cục
- $_SERVER — Mảng các biến máy chủ
- $_GET — Mảng các biến GET
- $_POST — Mảng các biến POST
- $_FILES — Mảng các tệp upload
- $_REQUEST — Mảng các biến Request (cả GET và POST)
- $_SESSION — Mảng các biến phiên
- $_ENV — Mảng các biến môi trường
- $_COOKIE — Mảng các biến Cookies

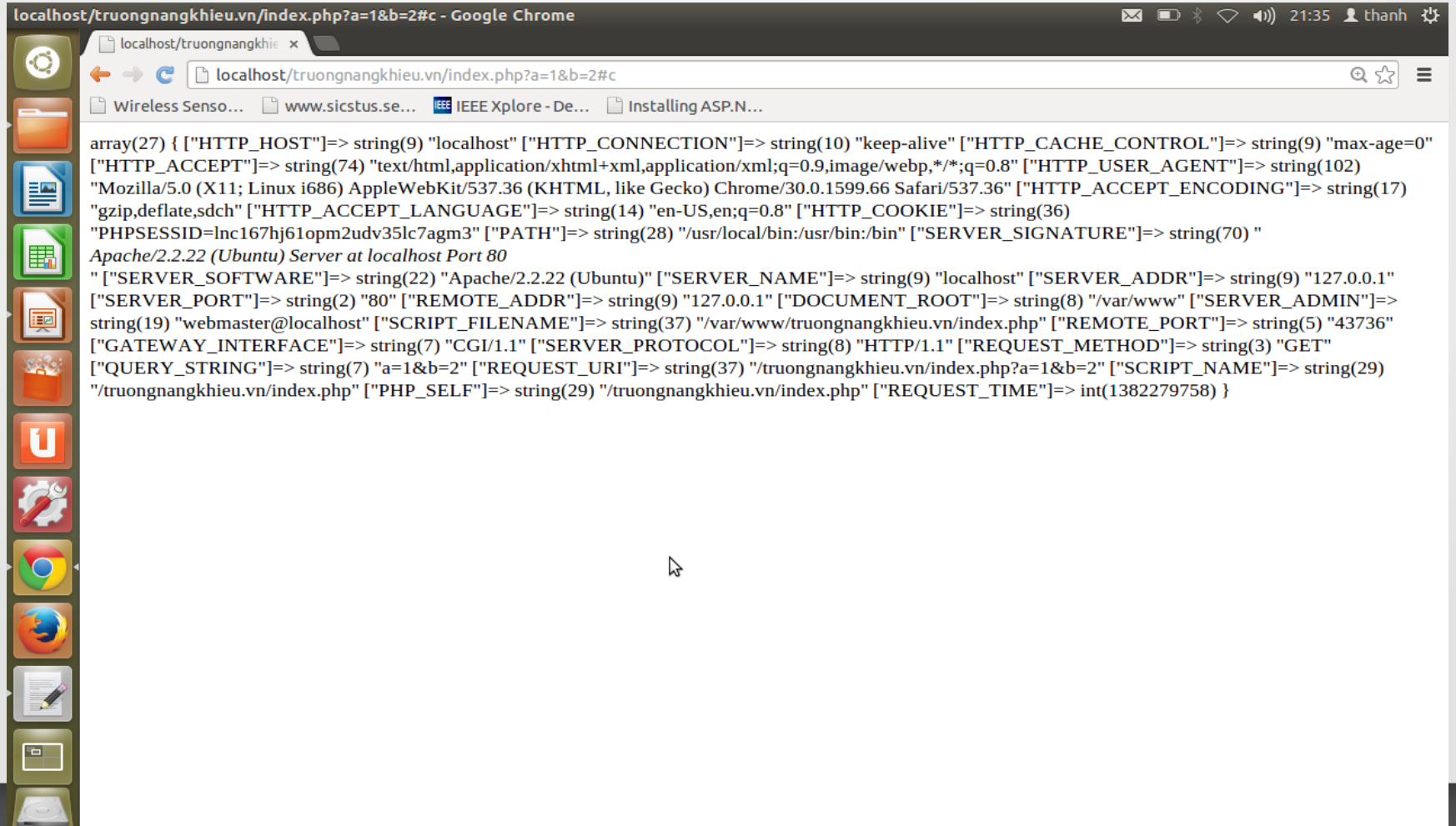
$GLOBALS

```
<?php    var_dump($GLOBALS);  ?>
```



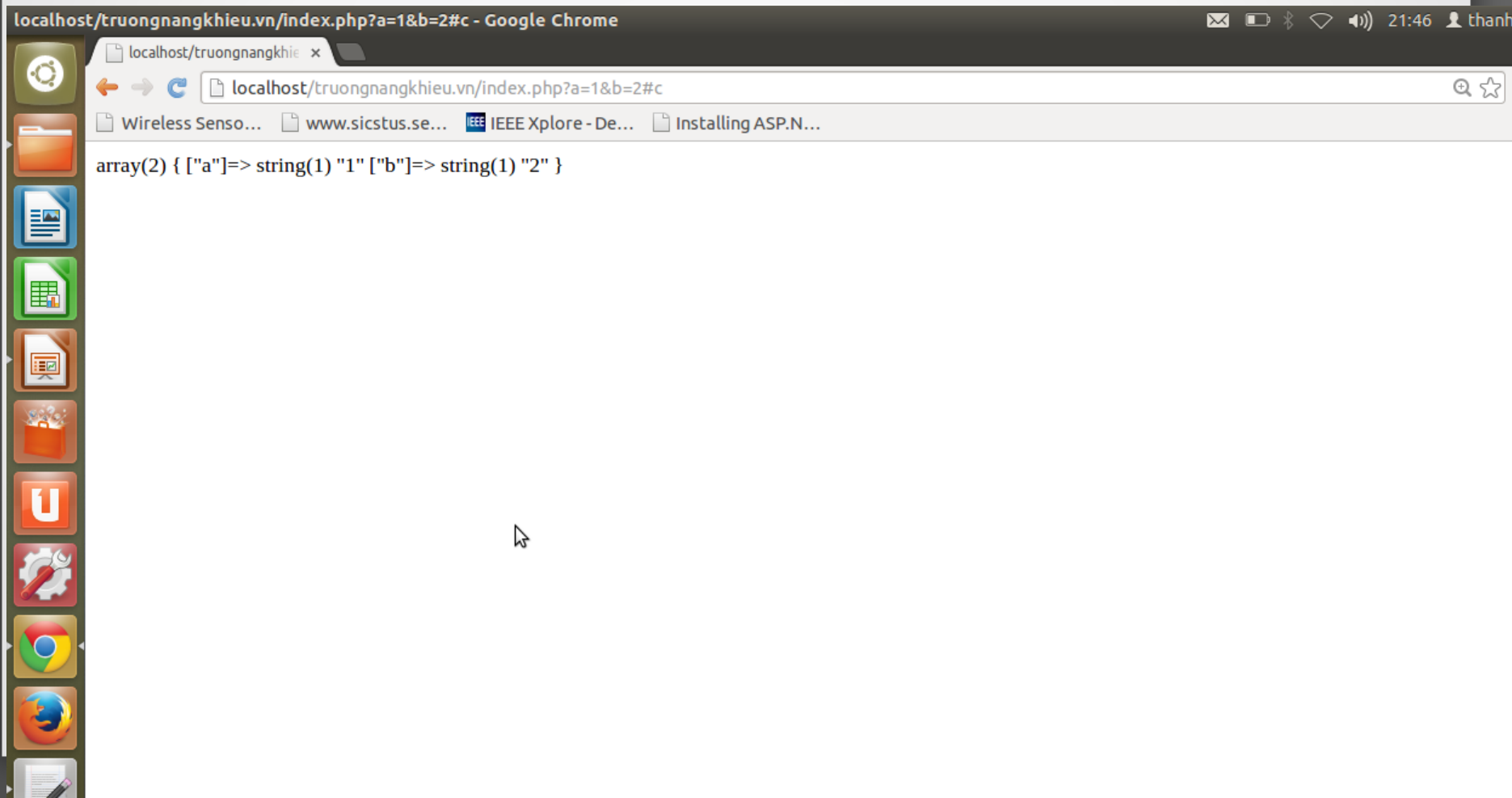
$_SERVER

<?php var_dump($_SERVER); ?>



$_GET

- <?php var_dump($_GET); ?>



Nhận tham số từ GET Request

E:\thanh\htdocs\labs\webapp-development\get.php - Notepad++

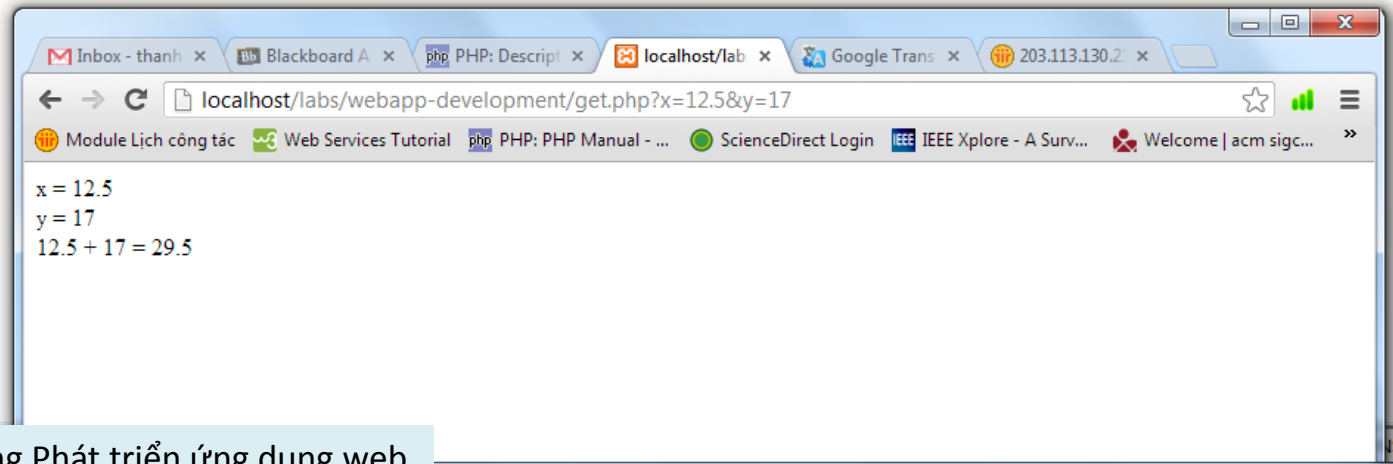
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

index.php | ns.php | ns1.php | ns3.php | first-example.php | new 10 | clone.php | get.php

```php
<?php
    if (isset($_GET["x"])) {
        echo "x = ".$_GET["x"];
    }

    if (isset($_GET["y"])) {
        echo "<br>y = ".$_GET["y"];
    }

    if (isset($_GET["x"]) && isset($_GET["x"])) {
        echo "<br>".$_GET["x"]." + ".$_GET["y"]." = ";
        echo (floatval($_GET["x"]) + floatval($_GET["y"]));
    }
?>
```



Nhận tham số từ POST Request

Firefox Web Browser 16:22

Open Save

```php
<?php
    //Đã đệ trình form
    if (isset($_POST["x"]) && isset($_POST["y"])) {
        //Bước 1: Kiểm tra các ràng buộc trên dữ liệu vào
        if (!is_numeric($_POST["x"])) { echo "x phải là số"; }
        else if (!is_numeric($_POST["y"])) { echo "y phải là số"; }
        else {
            //Bước 2: Giải bài toán
            $sum = floatval($_POST["x"]) + floatval($_POST["y"]);
            //Bước 3: Xuất kết quả
            echo $_POST["x"]." + ".$_POST["y"]." = ".$sum;
        }
    }
?>
<form method="post">
    x = <input type="text" name="x" value="<?php echo (isset($_POST["x"]) ? $_POST["x"] :""); ?>"> <br>
    y = <input type="text" name="y" value="<?php echo (isset($_POST["y"]) ? $_POST["y"] :""); ?>"> <br>
    <input type="submit" value="Submit">
</form>
```

Mozilla Firefox

http://localhost/lects/we

localhost/lects/webapp-development/php/sum2.php

Search

56 + 78 = 134

x = 56

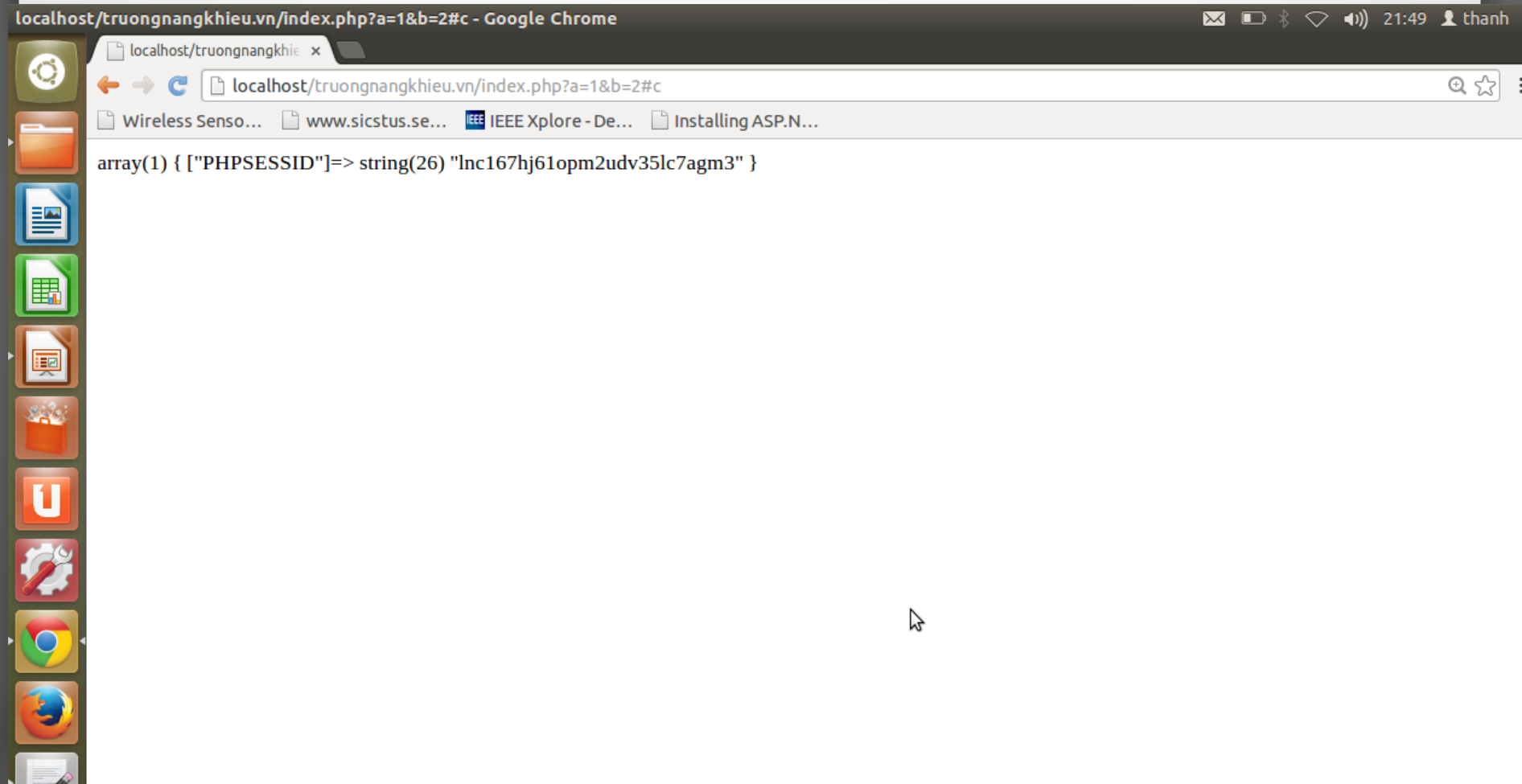
y = 78

Submit

INS

$_COOKIE

<?php var_dump($_COOKIE); ?>



Tạo tiêu đề gói HTTP Response

void **header** (string $string [, bool $replace = *true* [, int $http_response_code]])

- Ví dụ

```
header("HTTP/1.0 404 Not Found");
header("Location: http://www.example.com/"); /* Redi
rect browser */
header('WWW-Authenticate: Negotiate');
```

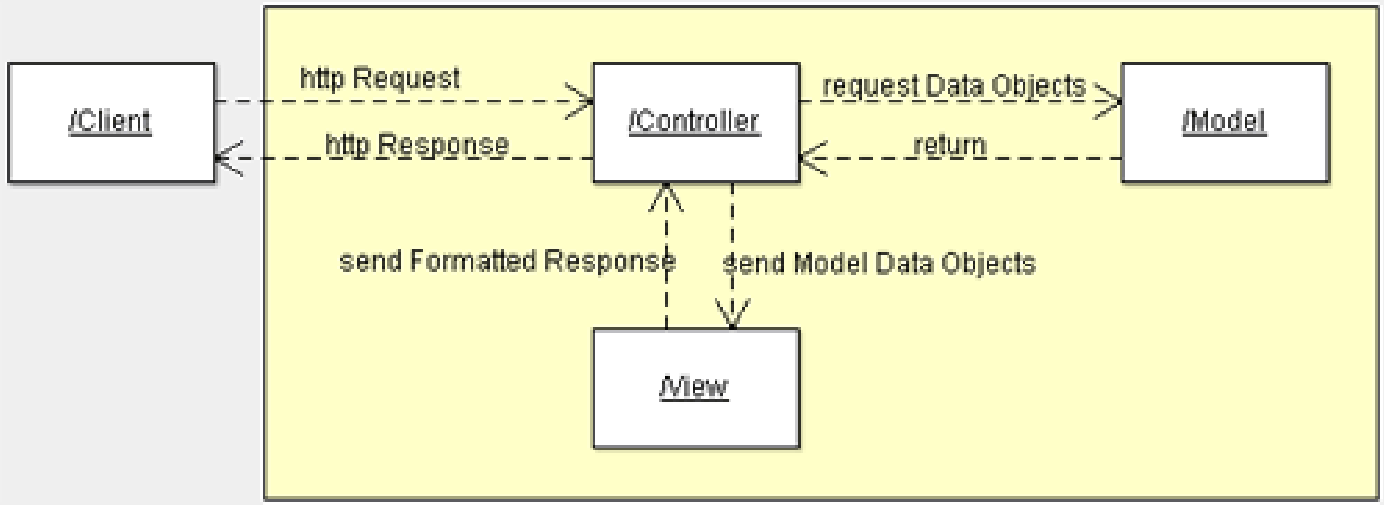
Các hàm khác

- Các hàm để đặt cookie, session, chuyển đổi biểu diễn địa chỉ ip, … (*xem ở các bài sau và trong tài liệu tham khảo*)

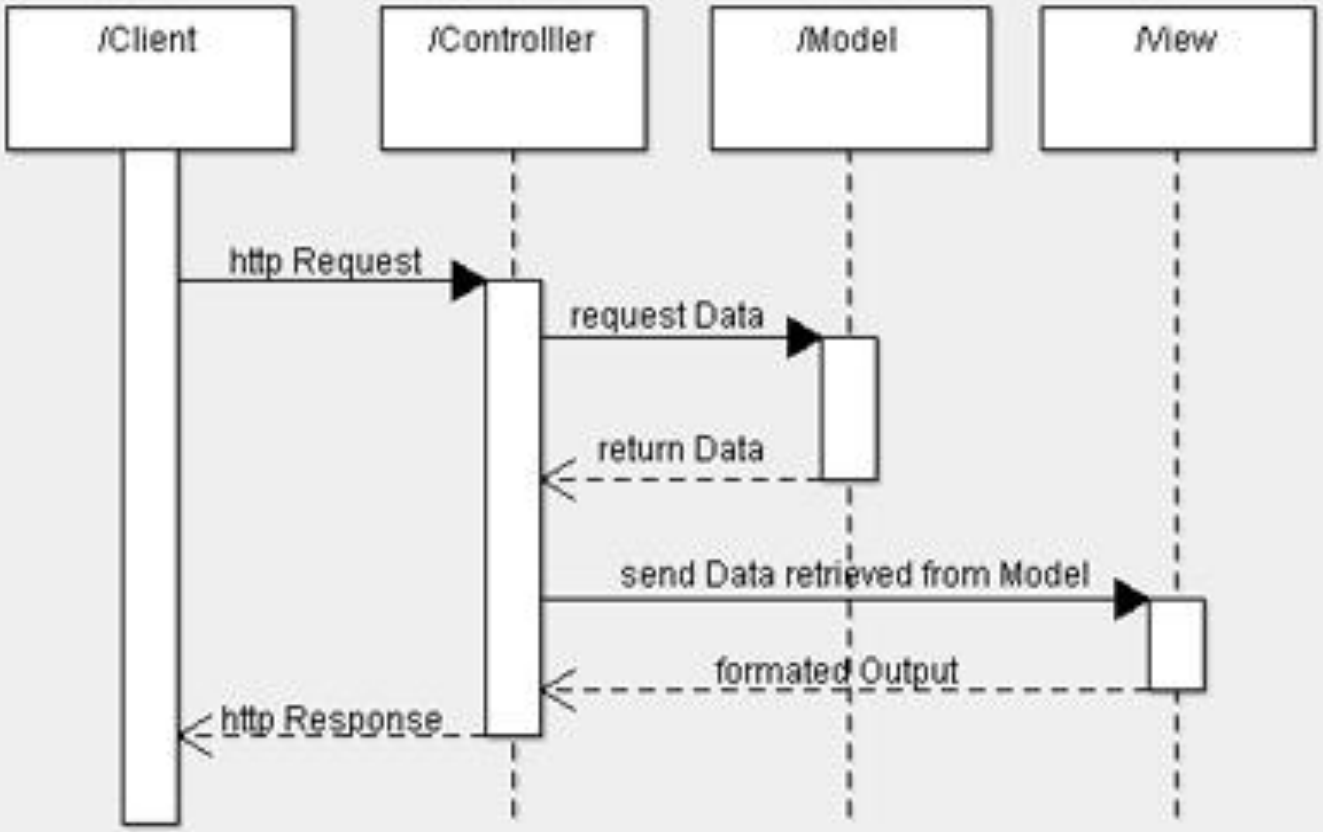
Mẫu thiết kế
Model – View - Controller

Model – View - Controller

- MVC là mẫu thiết kế được sử dụng rộng rãi cho ứng dụng web
  - **Model**: Xử lý logic của ứng dụng, nhận input, cho ra output
  - **View**: Format dữ liệu ra dưới định dạng cụ thể (HTML, JSON, BLOB, ...)
  - **Control**: Giao tiếp với client, điều phối model và view làm viêc



Model – View - Controller



Ví dụ: Tổng hai số

- Model:
  - Thành viên dữ liệu
    - x, y: Các số hạng
    - sum: Tổng các số hạng
  - Thành viên hàm
    - solve(): Giải bài toán
    - getSum(): Trả về nghiệm của bài toán

Ví dụ: Tổng hai số

```php
<?php
        /**
        *  model.php: Tính tổng hai số
        */
        class SumModel {
                //Input:
                private $x; //Số thứ nhất
                private $y; //Số thứ hai
                //Output:
                private $sum; //Tổng

                //Nhận dữ liệu vào
                public function __construct($x, $y) {
                        $this->x = $x;  $this->y = $y;
                }
                /**
                * Giải bài toán
                */
                public function solve() {  $this->sum = $this->x + $this->y; }
                /**
                * Trả kết quả
                */
                public function getSum() { return $this->sum; }
```

Ví dụ: Tổng hai số

- View: Hiển thị kết quả
  - Thành viên dữ liệu
    - x, y, ret: Các số hạng và tổng
    - render(): Tạo nội dung web từ dữ liệu

Ví dụ: Tổng hai số

```php
<?php
        /**
        *  view.php: Trình diễn kết quả tính tổng hai số
        */
        class SumView {
                private $x;   private $y;  private $ret;
                //Nhận dữ liệu vào và ra
                public function __construct($x, $y, $ret) {
                        $this->x = $x;  $this->y = $y; $this->ret = $ret;
                }
                /**
                * Tạo trang web từ dữ liệu thô
                */
                public function render() {
                        $html = "<!DOCTYPE html> ";
                        $html .= "<html>";
                        $html .= "<head>";
                        $html .= "<title>Tong hai so</title>";
                        $html .= "<meta charset=utf-8'>";
                        $html .= "</head>";
                        $html .= "<body>";
                        $html .= "<h1>Tổng hai số</h1>";
                        $html .= $this->x;
                        $html .= " + ";
                        $html .= $this->y;
                        $html .= " = ";
                        $html .= $this->ret;
                        $html .= "</body></html>";
                        return $html;
```

Ví dụ: Tổng hai số

- Controller: Nhận các tham số của người dùng, điều phối model và view hoạt động, trả kết quả cho người dùng
  - Thành viên dữ liệu
    - Không
  - Thành viên hàm
    - proc(): Thực hiện các công việc nêu trên

Ví dụ: Tổng hai số

```php
<?php
/**
*  control.php: Điều khiển chương trình tính tổng hai số
*/
require_once("model.php");
require_once("view.php");

class SumControl {
    public function proc() {
        //1. Nhận yêu cầu, kiểm tra các tham số

        if (isset($_GET["x"]) && isset($_GET["x"]) &&
           is_numeric($_GET["x"]) && is_numeric($_GET["y"])) {
            $x = $_GET["x"];
            $y = $_GET["y"];

            //2. Gọi model để xử lý nghiệp vụ
            $model = new SumModel($x, $y);
            $model->solve();
            $ret = $model->getSum(); //Kết quả xử lý nghiệp vụ

            //3. Gọi view để tạo nội dung
            $view = new SumView($x, $y, $ret);
            $html = $view->render();

            //4. Trả lời trình khách
            echo $html;
        } else {
            echo "Nhập x, y là các số. Ví dụ  ?x=3&y=-4";
```

Ví dụ: Tổng hai số

- Khai thác các thành phần MVC
- index.php

```
<?php
        require_once("control.php");
        $ctrl = new SumControl();
        $ctrl->proc();
```

Giao diện cấu phần hoặc JSON

Giao diện cấu phần

•View không trả về trang web đầy đủ mà chỉ trả về một đoạn nội dung cấu phần của trang web

```
<?php
        class SumView {
                ...
                public function render() {
                        $html = "<div> ";
                        $html .= "<h1>Tổng hai số</h1>";
                        $html .= $this->x;
                        $html .= " + ";
                        $html .= $this->y;
                        $html .= " = ";
                        $html .= $this->ret;
                        $html .= "</div>";
                        return $html;
                }
```

Giao diện cấu phần

•Sử dụng giao diện cấu phần

<!-- index.php -->

```
<!DOCTYPE html><html><head>
        <title>MVC 2</title>
        <meta charset="utf-8">
</head><body>
<?php
        require_once("control.php");
        $ctrl = new SumControl();
        $ctrl->proc();
?>
</body></html>
```

Trả về dữ liệu JSON

- View không cần nữa
- Model trả về dữ liệu JSON

```
<?php
        class SumModel {
                ...
                public function __construct($x, $y) { ... }
                public function solve() { ... }
                public function getSum() { ... }
                /**
                * Trả kết quả JSON
                */
                public function getSumJSON() {
                 $arr = array("x"=>$this->x, "y"=>$this->y, "ret"=>$this->sum);
                  return json_encode($arr);
                }
        }
```

Trả về dữ liệu JSON

- Controller chuyển dữ liệu JSON cho frontend

```
<?php
        require_once("model.php");
        require_once("view.php");

        class SumControl {
                public function proc() {
                        ...
                        $model->solve();
                        //Kết quả xử lý nghiệp vụ ở định dạng JSON
                        $json = $model->getSumJSON();
                        echo $json;
                        ...
                }
        }
```

Trả về dữ liệu JSON

•Frontend phân tích dữ liệu JSON và cập nhật giao diện

```
<!DOCTYPE html><html><head>
        <title>MVC 3</title> <meta charset="utf-8">
</head><body>
        <div>
        <h1>Tổng hai số</h1>
        <span id="x"></span> + <span id="y"></span> = <span id="ret"></span>
        </div>
        <script>
                <?php
                        require_once("control.php");
                        $ctrl = new SumControl();
                        echo "var json = JSON.parse('";
                        $ctrl->proc3();
                        echo "');";
                ?>
                document.getElementById("x").innerHTML = json.x;
                document.getElementById("y").innerHTML = json.y;
                document.getElementById("ret").innerHTML = json.ret;
        </script>
</body></html>
```

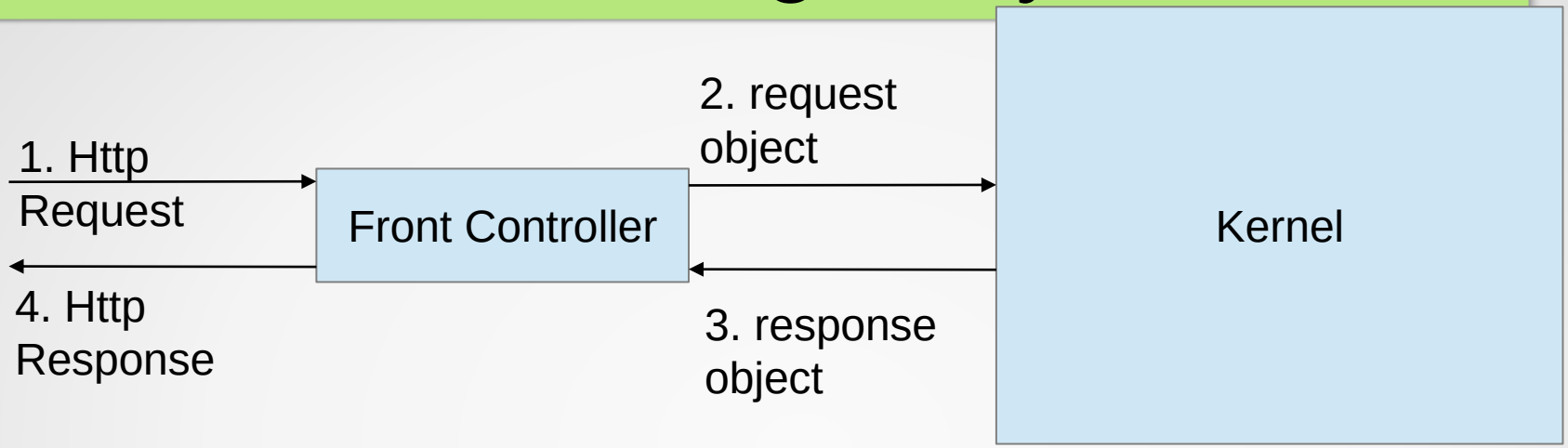
Laravel

https://laravel.com

Cấu trúc thư mục

- *app/Http/Controllers*
  chứa các tệp cài đặt các lớp điều khiển
- *resources/views*
  chứa các tệp cài đặt các lớp giao diện
- *app/Models*
  chứa các các tệp cài đặt các lớp mô hình
- *routes*
  chứa các tệp lưu trữ thông tin cho định tuyến URL
- *config*
  lưu các tệp cấu hình của ứng dụng,
- *database*
  lưu các tệp liên quan đến thao tác CSDL
- *public*
  lưu các tài nguyên khác như hình ảnh, JavaScript, CSS, …
- *storage*
  lưu các tệp session, cache và các tệp khác được sinh ra bởi khung làm việc
- *tests*
  chứa các kịch bản kiểm thử do người lập trình tạo lập
- *vendor*
  chứa các môđun do bên thứ ba cung cấp.

Luồng xử lý

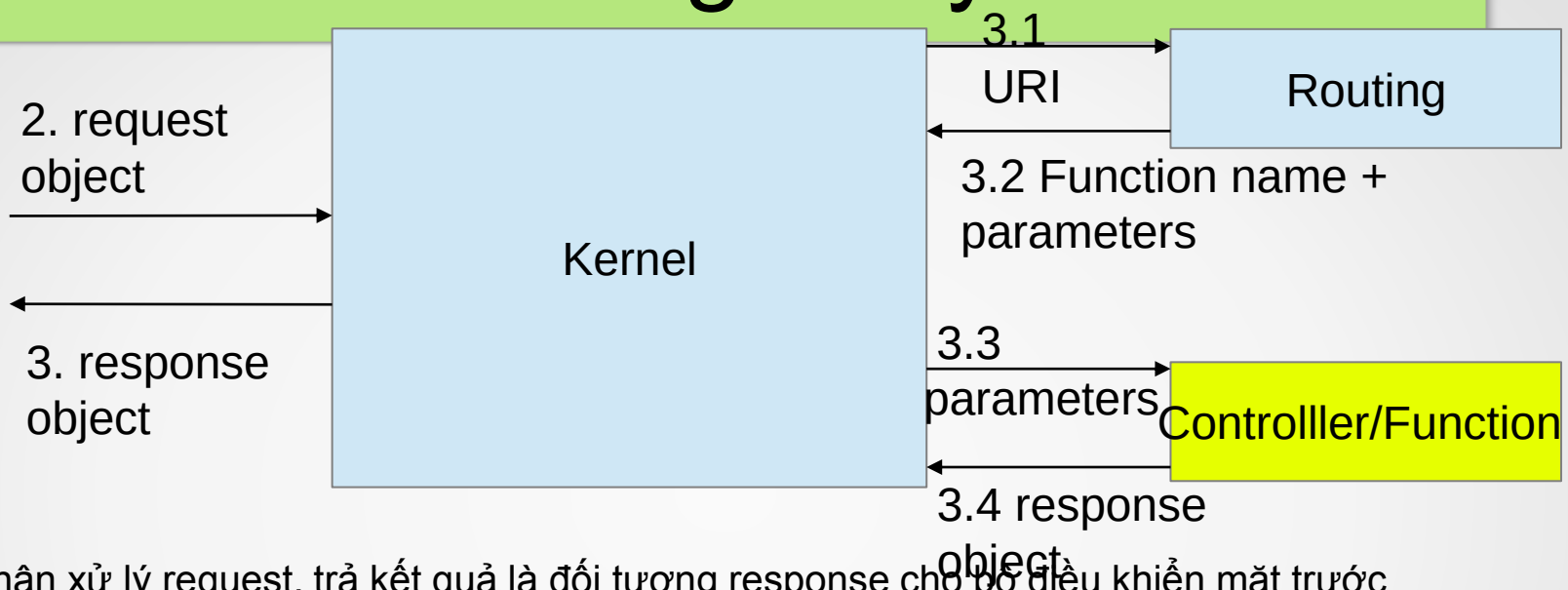


- Bộ điều khiển mặt trước (front controller) khởi động nhân ứng dụng (kernel), tiếp nhận Http Request, tạo đối tượng request và đưa Http Request vào đối tượng request được tạo, sau đó chuyển đối tượng request cho nhân
- Nhân xử lý request, trả kết quả là đối tượng response cho bộ điều khiển mặt trước
- Tại bộ điều khiển mặt trước, đối tượng response được chuyển thành Http Response rồi gửi cho client

Luồng xử lý

```
//Khởi động nhân ứng dụng
$kernel = $app->make(Illuminate\Contracts\Http\Kernel::class);
//Tạo đối tượng request nắm giữ Http Request
$request = Illuminate\Http\Request::capture();
//Chuyển đối tượng request cho nhân xử lý, nhận kết quả xử lý là đối tượng response
$response = $kernel->handle($request);
//Tạo và gửi Http Response cho client
$response->send();
//Tắt nhân
$kernel->terminate($request, $response);
```

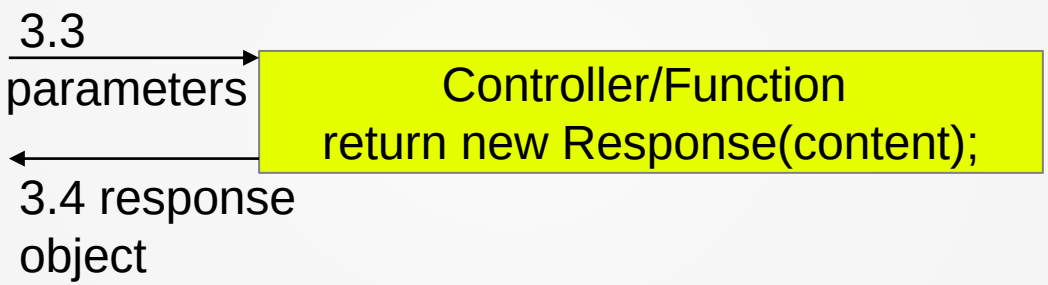
Luồng xử lý



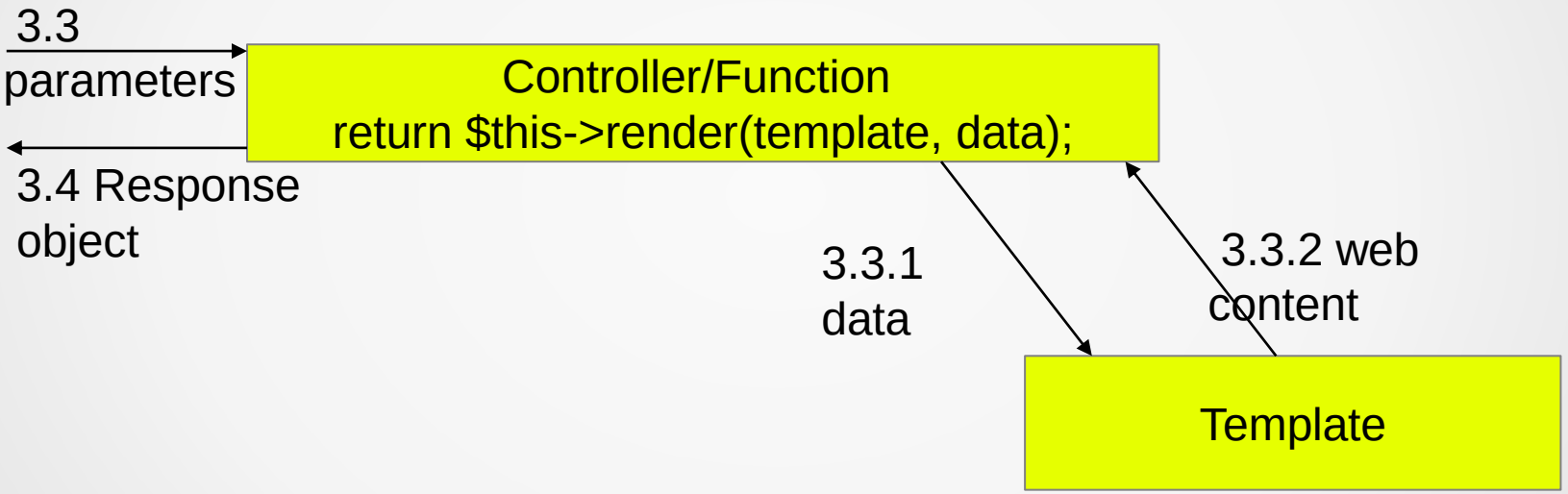
- Nhân xử lý request, trả kết quả là đối tượng response cho bộ điều khiển mặt trước
  - **Nhân chuyển URI của request cho hệ thống định tuyến (routing)**
  - **Hệ thống định tuyến xác định hàm PHP (còn được gọi là bộ điều khiển (controller) hay hành động (action)) nào sẽ được gọi**
  - **Hàm PHP được gọi thực hiện thông dịch, xử lý request và tạo đối tượng response**
  - **Đối tượng response được trả về cho nhân và nhân chuyển cho bộ điều khiển mặt trước**
-

Tạo “trang web” = viết hàm xử lý request và tạo response + ánh xạ URL tới hàm

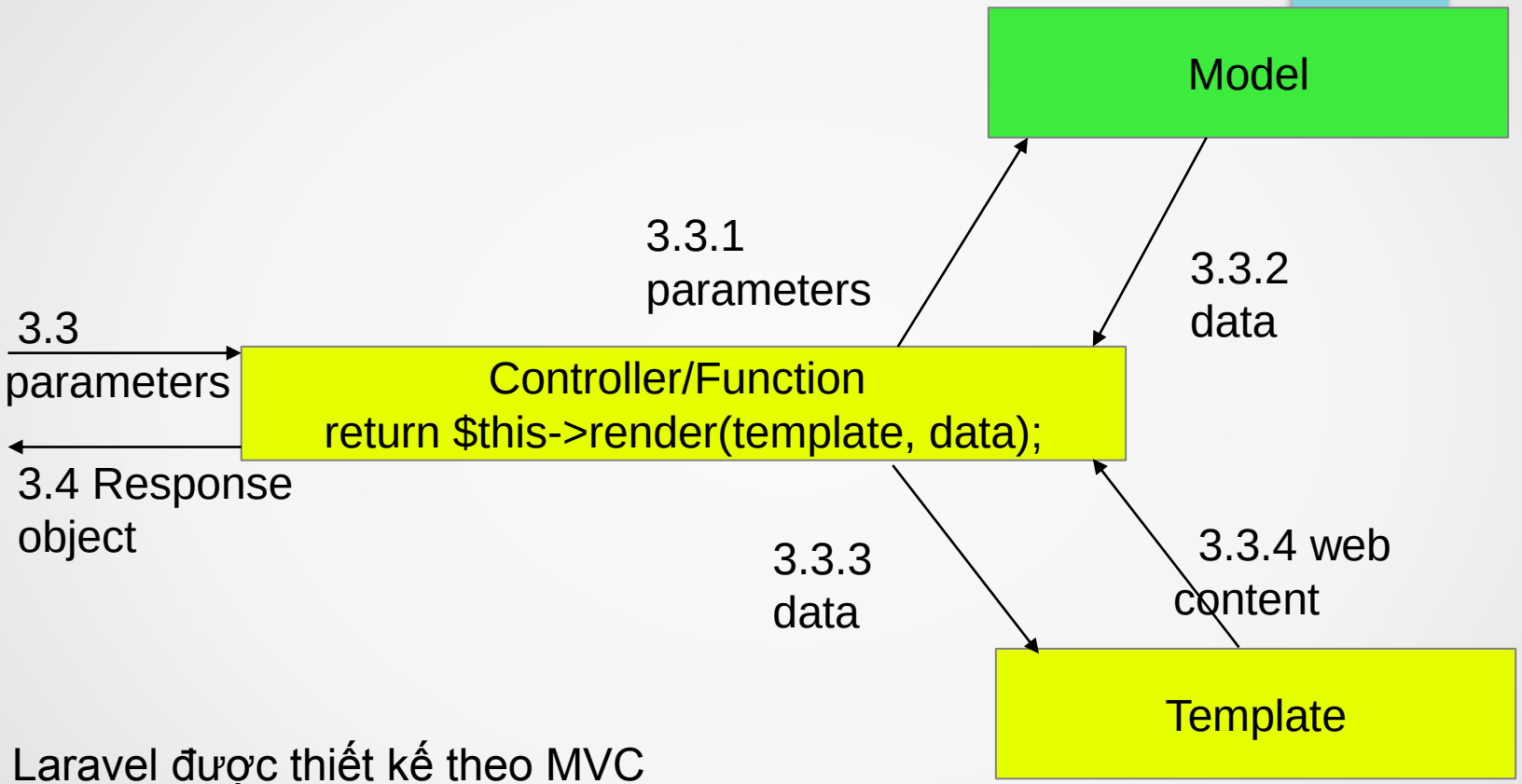
Controller trực tiếp tạo Response



Controller sử dụng template/view để tạo Response



Sử dụng model



Laravel được thiết kế theo MVC
nhưng
không bắt buộc và không hỗ trợ
model.
Người phát triển tự tạo model.

Thiết lập thông tin định tuyến URL

- *Route::method($uri, $callback);*

  method = HTTP verd: get, post, put, delete, ...

- *Ví dụ*

```
Route::get('/greeting', function() { return view('welcome'); });

Route::get('readers/{readerId}', function ($rid) {return 'Độc giả: '.$rid;
    });

Route::get('posts/{postId}/comments/{commentId}', function($pid, $cid
    ) { return 'Bạn đang đọc bình luận '.$cid.' ở bài viết'.$pid; });

Route::match(['get', 'post'], '/help', function () { return view('help');    });

Route::any('/licence', function () { return "Apache Licence";  });

Route::redirect('/here', '/there', 301);
```

Xây dựng lớp điều khiển

- php artisan make:controller <ControllerName>
- Ví dụ
  php artisan make:controller BookController

```
<?php
  // Tệp app/Http/Controllers/BookConroller.php
  namespace App\Http\Controllers;
  use App\Http\Controllers\Controller;

  class BookController extends Controller
  {
    public function show($bid)
    {
      return "Xem thông tin cuốn sách $bid";
    }
  }
```

Đọc giá trị của tham số trong yêu cầu HTTP

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;

class BookController extends Controller {
  //
  public function store(Request $request) {
    $title = $request->input('title');
    $author = $request->input('authors');
    //
  }
  public function update(Request $request, $bid) {
    $title = $request->input('title');
    $author = $request->input('authors');
    //
  }
}
```

Trả về text hoặc JSON

```
class BookController extends Controller {
  //  Trả về text
  public function index() {
    return 'Sách: Gió Thu, Sóng Sánh, Chiều Hồng';
  }
  // Trả về JSON
  public function index2() {
    return ['Gió Thu', 'Sóng Sánh', 'Chiều Hồng'];
  }
}
```

Trả về đối tượng response

```php
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Response;

class BookController extends Controller {
//
  public function index() {
    $content = ['Gió Thu', 'Sóng Sánh', 'Chiều Hồng'];
    return response($content) ;
  }
}
```

Thêm tiêu đề cho HTTP response

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Response;

class BookController extends Controller {
//
  public function index() {
    $content = ['Gió Thu', 'Sóng Sánh', 'Chiều Hồng'];
    return response($content)
              ->withHeaders([
                'Content-Type' => 'application/json',
                'Set-Cookie' => 'view=list;HttpOnly'
              ]);
    }
}
```

Tạo model

php artisian make:model Models\Book

```
<?php
  // Tệp app/Models/Book.php
  namespace App\Models;
  use Illuminate\Database\Eloquent\Model;

  class Book extends Model {
    protected $table='Sach';
  }
```

Truyền dữ liệu cho view

```
<?php
  // Tệp app/Http/Controllers/BookConroller.php
  namespace App\Http\Controllers;
  use App\Book;
  use App\Http\Controllers\Controller;
  class BookController extends Controller {
    public function show($id) {
      return view('showbook', ['book' => Book::findOr
Fail($id)]);
    }
  }
```

Truyền dữ liệu cho view

```
<!-- resources/views/layouts/showbook.blade.php -->
@extends('layouts.master)

@section('header')
  <p>Thông tin sách</p>
@endsection

@section('content')
  <h2>{{ $book->tieude }} </h2>
  Các tác giả:
  @foreach ($book->authors as $author)
    {{ $author->hoten }} <br>
  @endforeach
  Năm xuất bản: {{ $book->namxb }}
@endsection
```

View master

```
<!-- resources/views/layouts/master.blade.php -->
<!DOCTYPE html><html><head>
  <title>ABC website</title>
  <meta charset="utf-8">
</head><body>
  <div class="container">
    <div>@yield('header')</div>
    <div class="row">
      <div class="col-md-3">
        @section('leftmenu')
          Left menu
        @show
      </div>
      <div class="col-md-9">@yield('content')</div>
    </div>
  </div>
</body></html>
```

Tiếp theo

Định tuyến URL