TVET CERTIFICATE V in SOFTWARE DEVELOPMENT

WEB APPLICATION DEVELOPMENT

SFDWA501 Develop a web application

Competence

Credits: 12 **Learning hours: 120**

Sector: ICT

Sub-sector: Software Development

Module Note Issue date: June, 2020

Purpose statement

This module describes the skills and knowledge required to scope the design of website. It applies to individuals working as web designers and web developers, who apply a wide range of knowledge and skills for basics web development.

Table of Contents

Total Number of Pages: 135

Learning Unit 1 – Apply PHP Fundamentals

LO 1.1 – Define PHP language and its basics

- **Topic 1: Introduction to PHP as a scripting language**

  - **What does PHP stand for?**

PHP is a server side scripting language. That is used to develop Static websites or Dynamic websites or Web applications. PHP stands for **Hypertext Pre-processor**, that earlier stood for Personal Home Pages. PHP scripts can only be interpreted on a server that has PHP installed. The client computers accessing the PHP scripts require a web browser only. A PHP file contains PHP tags and ends with the extension ".php".

  - **Purpose of PHP**

PHP is a general-**purpose** scripting language that is especially suited to server-side web development, in which case **PHP** generally runs on a web server.

PHP is mainly focused on server-side scripting, so you can do anything any other CGI program can do, such as collect **form** data, generate dynamic page content, or send and receive cookies. But PHP can do much more. There are three main areas where PHP scripts are used. One of the important advantages of PHP is that it is Open Source. Therefore, PHP is readily available and is entirely free. In contrast to other scripting languages used for web development which requires the user to **pay** for the support files, PHP is open to everyone, anytime and **anywhere**.

  - **PHP files**

File extension and Tags In order for the **server** to **identify** our **PHP files** and **scripts**, we must **save** the **file** with the **".php" extension**. Older PHP file extensions include

- .phtml
- .php3
- .php4
- .php5
- .phps

PHP was designed to work with HTML, and as such, it can be embedded into the HTML code.

<HTML> <PHP CODE> </HMTL>

You can create PHP files without any html tags and that is called Pure PHP file. The server interprets

the PHP code and outputs the results as HTML code to the web browsers. In order for the server to identify the PHP code from the HTML code, we must always enclose the PHP code in PHP tags.
A PHP tag starts with the less than symbol followed by the question mark and then the words php".
PHP is a case sensitive language, "VAR" is not the same as "var". The PHP tags themselves are not case-sensitive, but it is strongly recommended that we use lower case letter. The code below illustrates the above point.

- **PHP program structure**

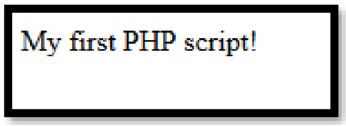
Basic PHP Syntax
A PHP script can be placed anywhere in the document. A PHP script starts with **<?php** and ends with **?>**:

```
<?php
// PHP code goes here
?>
```

Example

```
<!DOCTYPE html>
<html>
<body>

<?php
echo "My first PHP script!";
?>
</body>
</html>
```

Output:



```
My first PHP script!
```

- **PHP code is executed on the server.**

The default file extension for PHP files is ".php". A PHP file normally contains HTML tags, and some PHP scripting code.
Below, we have an example of a simple PHP file, with a PHP script that uses a built-in PHP function "echo" to output the text "Hello World!" on a web page:

Example:

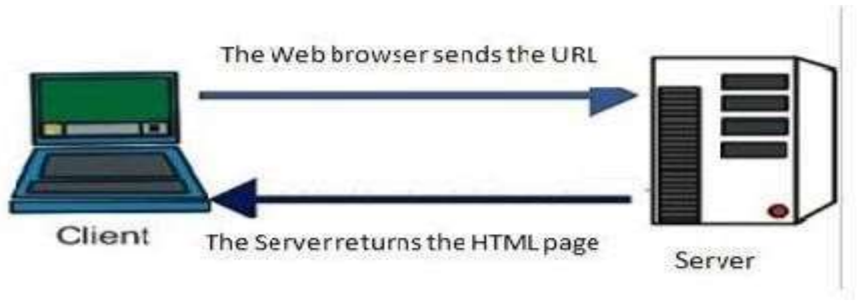
```
<!DOCTYPE html>
<html>
<body>
<h1>My first PHP page</h1>
<?php
echo "Hello World!";
?>
</body>
</html>
```

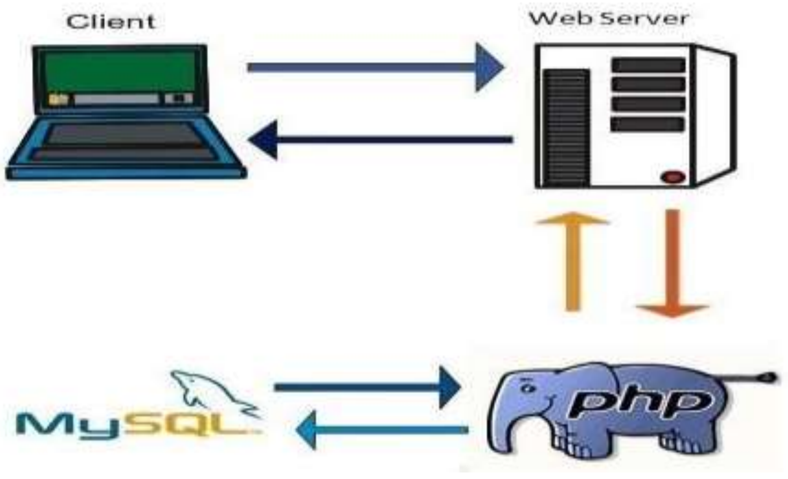
Note: PHP statements end with a semicolon (;).

- Topic 2. Difference between scripting and programming languages

- **Server-side processes**

Server-side processing is used to interact with permanent **storage** like databases or files. The server will also render pages to the client and process user input. Server-side processing happens when a page is first requested and when pages are posted back to the server.


The Web browser sends the URL
Client
The Server returns the HTML page
Server



Your browser sends the address which you typed:

- The Web server seeks in its tree structure if the file exists, and if this one carries an extension recognized like a PHP application (PHP, PHP3, PHP4 for example).If it is the case, the Web server transmits this file to PHP.
- PHP analyses the file, i.e. it will analyse and execute the PHP code which is between the <?php and ? > mark-up. If this code contains queries towards a MySQL database, PHP sends SQL request. The database returns wanted information to the script which can exploit them (to display them for example).
- PHP continues to analyse the page, then turns over the file deprived of PHP code to the Web server.
- The Web server thus returns to the browser a file not containing more PHP therefore containing only HTML, the web browser interprets it and displays it.

- **Interpreter**

A local PHP interpreter is a PHP engine installed on your computer.

Installing and configuring XAMPP

XAMPP is a cross-platform package consisting of an Apache HTTP server, MySQL database, PHP interpreter, and Perl interpreter. The word "XAMPP" is an acronym, where "X" stands for "cross", meaning "cross-platform", and the other letters stand for the package components.

XAMPP is a reliable and fast way to set up environment for PHP programming. It provides all the components required for developing, running, debugging, and unit testing of PHP applications. XAMPP is a good alternative to installing and configuring a Web server, a PHP engine, a database server, and a debug engine separately. All you need to do to start developing is download XAMPP, run XAMPP installer.exe, and start the components using the XAMPP control panel.

- **Compiler**

A program that converts instructions into a machine-code or lower-level form so that they can be read and executed by a computer.

A compiler is a computer program that translates computer code written in one programming language into another programming language. The name compiler is primarily used for programs that translate source code from a high-level programming language to a lower level language to create an executable program.

The **PHP** language is **interpreted**. The binary that lets you interpret **PHP** is compiled, but what you write is **interpreted**. Both. **PHP** is compiled down to an intermediate bytecode that is then **interpreted** by the runtime engine.

- **Inside HTML files**

Php is a programming style language used to create pages that are processed and served from the server. **Php files** can always read and display **HTML** code, but **HTML** does not automatically parse **php** code. To do so, you will need to make adjustments to your .htaccess **file**.

By default, you **can**'t **use PHP in HTML** pages. If you only have **php** code in one **html** file but have multiple other files that only contain **html** code, you **can** add the following to your .htaccess file so it **will** only serve that particular file as **php**.

LO 1.2 – Describe PHP Syntax, Data types, Variables, Operators and Arrays.

- Topic 1: Identification of PHP Syntax rules

- **Primitive Data types**

The values assigned to a PHP variable may be of different data types including simple string and numeric types to more complex data types like arrays and objects.

PHP supports total eight primitive data types: Integer, Floating point number or Float, String, Booleans, Array, Object, resource and NULL. These data types are used to construct variables. Now let's discuss each one of them in detail.

PHP Integers

Integers are whole numbers, without a decimal point (..., -2, -1, 0, 1, 2, ...). Integers can be specified in decimal (base 10), hexadecimal (base 16 - prefixed with 0x) or octal (base 8 - prefixed with 0) notation, optionally preceded by a sign (- or +).

Example

```
<?php
$a = 123; // decimal number
```

```
var_dump($a); echo "<br>";

$b = -123; // a negative number
var_dump($b); echo "<br>";
$c = 0x1A; // hexadecimal number
var_dump($c); echo "<br>";
$d = 0123; // octal
number
var_dump($d); ?>
```

Note: Since PHP 5.4+ you can also specify integers in binary (base 2) notation. To use binary notation precede the number with 0b (e.g. `$var = 0b11111111;`).

PHP Strings

Strings are sequences of characters, where every character is the same as a byte. A string can hold letters, numbers, and special characters and it can be as large as up to 2GB (2147483647 bytes maximum). The simplest way to specify a string is to enclose it in single quotes (e.g. 'Hello world!'), however you can also use double quotes ("Hello world!").

Example

```
<?php
$a = 'Hello world!'; echo
$a; echo "<br>";
$b = "Hello
world!"; echo $b;
echo "<br>";
$c = 'Stay here, I\'ll be back.'; echo $c;
?>
```

PHP Floating Point Numbers or Doubles

Floating point numbers (also known as "floats", "doubles", or "real numbers") are decimal or fractional numbers, like demonstrated in the example below.

Example

```
<?php
$a =
1.234;
```

```
var_dump($
a); echo
"<br>"; $b
= 10.2e3;
var_dump(
$b); echo
"<br>";
$c = 4E-10; var_dump($c);
?>
```

HP Booleans

Booleans are like a switch it has only two possible values either $_1$ (true) or $_0$ (false).

Example

```
<?php
// Assign the value TRUE to a variable
$show_error = true; var_dump($show_error);
?>
```

PHP Arrays

An array is a variable that can hold more than one value at a time. It is useful to aggregate a series of related items together, for example a set of country or city names.

An array is formally defined as an indexed collection of data values. Each index (also known as the key) of an array is unique and references a corresponding value.

Example

```
<?php
$colors = array("Red", "Green", "Blue");
var_dump($colors); echo "<br>";
$color_codes =
   array( "Red" =>
   "#ff0000",
   "Green" =>
   "#00ff00", "Blue" =>
   "#0000ff"
);
var_dump($color_codes
); ?> PHP Objects
```

An object is a data type that not only allows storing data but also information on, how to process that data. An object is a specific instance of a class which serve as templates for objects. Objects are created based on this template via the new keyword.

Every object has properties and methods corresponding to those of its parent class. Every object instance is completely independent, with its own properties and methods, and can thus be manipulated independently of other objects of the same class.
Here's a simple example of a class definition followed by the object creation.

Example

```
<?php
// Class definition
class greeting{ //
properties
   public $str = "Hello World!";
   // methods
   function show_greeting(){         return
$this->str;
   }
}
// Create object from class
$message = new greeting; var_dump($message);
 ?>
```

Tip: The data elements stored within an object are referred to as its properties and the information, or code which describing how to process the data is called the methods of the object.

PHP NULL

The special NULL value is used to represent empty variables in PHP. A variable of type NULL is a variable without any data. NULL is the only possible value of type null.

Example

```
<?php
$a =
NULL;
var_dump($a); echo "<br>";
$b = "Hello World!";
```

```
$b = NULL;
var_dump($b); ?>
```

When a variable is created without a value in PHP like `$var;` it is automatically assigned a value of null. Many novice PHP developers mistakenly considered both `$var1 = NULL;` and `$var2 = "";` are same, but this is not true. Both variables are different — the `$var1` has null value while `$var2` indicates no value assigned to it.

PHP Resources

A resource is a special variable, holding a reference to an external resource. Resource variables typically hold special handlers to opened files and database connections.

Example

```
<?php
// Open a file for reading $handle =
fopen("note.txt", "r");
var_dump($handle); echo "<br>";
// Connect to MySQL database server with default setting
$link = mysql_connect("localhost", "root", "");
var_dump($link);
?>
```

- **Variable declaration**

Variables are "containers" for storing information. Creating (Declaring) PHP Variables.

In PHP, a variable starts with the $ sign, followed by the name of the variable:

Example

```
<?php
$txt = "Hello world!";
$x = 5;
$y = 10.5;
?>
```

After the execution of the statements above, the variable **$txt** will hold the value **Hello world!**, the variable **$x** will hold the value **5**, and the variable **$y** will hold the value **10.5**.

Note: 1. When you assign a text value to a variable, put quotes around the value.

2. Unlike other programming languages, PHP has no command for declaring a variable. It is created the moment you first assign a value to it.

A variable can have a short name (like x and y) or a more descriptive name (age, carname, total_volume).

- **Rules for PHP variables:**
    - ➢ A variable start with the $ sign, followed by the name of the variable
    - ➢ A variable name must start with a letter or the underscore character
    - ➢ A variable name cannot start with a number
    - ➢ A variable name can only contain alpha-numeric characters and underscores (A-z, 0-9, and _)
    - ➢ Variable names are case-sensitive ($age and $AGE are two different variables) Output Variables

The PHP **echo** statement is often used to output data to the screen. The following example will show how to output text and a variable:

Example

```
<?php
$txt = "W3Schools.com"; echo "I love $txt!";
?>
```

The following example will produce the same output as the example above:

Example

```
<?php
$txt = "W3Schools.com"; echo "I love " . $txt . "!";
?>
```

The following example will output the sum of two variables:

Example

```
<?php
$x = 5; $y = 4;
echo $x + $y;
?>
```

Note: You will learn more about the echo statement and how to output data to the screen in the next chapter.

PHP is a Loosely Typed Language. In the example above, notice that we did not have to tell PHP which data type the variable is. PHP automatically converts the variable to the correct data type, depending on its value.

- PHP Variables Scope

In PHP, variables can be declared anywhere in the script. The scope of a variable is the part of the script where the variable can be referenced/used. PHP has three different variable scopes:

- Local
- Global
- Static

Global and Local Scope

A variable declared **outside** a function has a GLOBAL SCOPE and can only be accessed outside a function:

Example

```
<?php
$x = 5; // global scope
function myTest() {
   // using x inside this function will generate an error
   echo "<p>Variable x inside function is: $x</p>";
}
myTest();
echo "<p>Variable x outside function is:
$x</p>"; ?>
```

A variable declared **within** a function has a LOCAL SCOPE and can only be accessed within that function:

Example

```
<?php
function myTest() {
   $x = 5; // local scope
   echo "<p>Variable x inside function is: $x</p>";
}
myTest();
// using x outside the function will generate an error
echo "<p>Variable x outside function is: $x</p>";
?>
```

You can have local variables with the same name in different functions, because local variables are only recognized by the function in which they are declared.

PHP The global Keyword

The **global** keyword is used to access a global variable from within a function. To do this, use the global keyword before the variables (inside the function):

Example

```
<?php
$x = 5;
$y = 10;
function yTest()
{ global $x, $y;
   $y = $x + $y;
}
myTest();
echo $y; // outputs 15
?>
```

PHP also stores all global variables in an array called $GLOBALS[*index*]. The *index* holds the name of the variable. This array is also accessible from within functions and can be used to update global variables directly.

The example above can be rewritten like this:

Example

```
<?php
$x = 5;
$y = 10;
function myTest() {
   $GLOBALS['y'] = $GLOBALS['x'] + $GLOBALS['y'];
}
myTest();
echo $y; // outputs 15
?>
```

PHP The static Keyword

Normally, when a function is completed/executed, all of its variables are deleted. However, sometimes we want a local variable NOT to be deleted. We need it for a further job. To do this, use the static keyword when you first declare the variable:

Example

```
<?php function myTest() { static $x = 0; echo $x;
   $x++;
}
myTest();
myTest();
myTest(); ?>
```

Then, each time the function is called, that variable will still have the information it contained from the last time the function was called.

Note: The variable is still local to the function.

* Constants

Constants are like variables except that once they are defined they cannot be changed or undefined. A constant is an identifier (name) for a simple value. The value cannot be changed during the script. A valid constant name starts with a letter or underscore (no $ sign before the constant name).

Note: Unlike variables, constants are automatically global across the entire script. Create a PHP Constant. To create a constant, use the define() function.

Syntax define(*name*, *value*, *case- insensitive*)

Parameters:

- *name*: Specifies the name of the constant
- *value*: Specifies the value of the constant
- *case-insensitive*: Specifies whether the constant name should be case-insensitive.

  Default is falseThe example below creates a constant with a **case-sensitive** name:

Example

```
<?php
define("GREETING", "Welcome to
W3Schools.com!"); echo GREETING;
?>
```

The example below creates a constant with a **case-insensitive** name:

Example

```
<?php
define("GREETING", "Welcome to W3Schools.com!",
true); echo greeting;
?>
```

Constants are Global

Constants are automatically global and can be used across the entire script. The example below uses a constant inside a function, even if it is defined outside the function:

Example

```
<?php
define("GREETING", "Welcome to W3Schools.com!");
function
   myTest() {
   echo
   GREETING;
}
myTest();
?>
```

- **Operators**

Operators are used to perform operations on variables and values. PHP divides the operators in the following groups:

- Arithmetic operators
- Assignment operators
- Comparison operators
- Increment/Decrement operators
- Logical operators
- String operators
- Array operators
- Conditional Operator

PHP Arithmetic Operators

The PHP arithmetic operators are used with numeric values to perform common arithmetical operations, such as addition, subtraction, multiplication etc.

Operator	Name	Example	Result
+	Addition	$x + $y	Sum of $x and $y
-	Subtraction	$x - $y	Difference of $x and $y
*	Multiplication	$x * $y	Product of $x and $y
/	Division	$x / $y	Quotient of $x and $y
%	Modulus	$x % $y	Remainder of $x divided by $y
**	Exponentiation	$x ** $y	Result of raising $x to the $y'th power (Introduced in PHP 5.6)

PHP Assignment Operators

The PHP assignment operators are used with numeric values to write a value to a variable. The basic assignment operator in PHP is "=". It means that the left operand gets set to the value of the assignment expression on the right.

The left operand gets set to the value of the expression on x = y x = y the

right x += y x = x + y Addition

x -= y x = x - y

Subtraction x *= y x = x

* y

Multiplication x /= y x = x

/ y Division x %= y x =

x

% y Modulus

PHP Comparison Operators

The PHP comparison operators are used to compare two values (number or string):

Operator	Name	Example	Result
==	Equal	$x == $y	Returns true if $x is equal to $y
===	Identical	$x === $y	Returns true if $x is equal to $y, and they are of the same type
!=	Not equal	$x != $y	Returns true if $x is not equal to $y
<>	Not equal	$x <> $y	Returns true if $x is not equal to $y

!==	Not identical	$x !== $y	Returns true if $x is not equal to $y, or they are not of the same type
>	Greater than	$x > $y	Returns true if $x is greater than $y
<	Less than	$x < $y	Returns true if $x is less than $y
>=	Greater than or equal to	$x >= $y	Returns true if $x is greater than or equal to $y
<=	Less than or equal to	$x <= $y	Returns true if $x is less than or equal to $y

PHP Increment / Decrement Operators

The PHP increment operators are used to increment a variable's value. The PHP decrement operators are used to decrement a variable's value.

Operator	**Name**	**Description**	**Show it**
++$x	Pre-increment	Increments $x by one,	then returns $x
$x++	Post-increment	Returns $x,	then increments $x by one
--$x	Pre-decrement	Decrements $x by one,	then returns $x
$x--	Post-decrement	Returns $x,	then decrements $x by one

PHP Logical Operators

The PHP logical operators are used to combine conditional statements.

Operator	**Name**	**Example**	**Result**
and	And	$x and $y	True if both $x and $y are true
or	Or	$x or $y	True if either $x or $y is true
xor	Xor	$x xor $y	True if either $x or $y is true, but not both
&&	And	$x && $y	True if both $x and $y are true
||	Or	$x || $y	True if either $x or $y is true
!	Not	!$x	True if $x is not true

PHP String Operators

PHP has two operators that are specially designed for strings.

Operator	Name	Example	Result	Show it
.	Concatenation	$txt1 . $txt2	Concatenation of $txt1 and $txt2	.= Concatenation assignment $txt1 .= $txt2 Appends $txt2 to $txt1

PHP Array Operators

The PHP array operators are used to compare arrays.

Operator	Name	Example	Result
+	Union	$x + $y	Union of $x and $y
==	Equality	$x == $y	Returns true if $x and $y have the same key/value pairs
===	Identity	$x ===$y	Returns true if $x and $y have the same key/value pairs in the same order and of the same types
!=	Inequality	$x != $y	Returns true if $x is not equal to $y
< >	Inequality	$x < >$y	Returns true if $x is not equal to $y
!==	Non- identity	$x !== $y	Returns true if $x is not identical to $y

Conditional /ternary Operator

The ternary operator allows us to simplify some PHP conditional statements Ex1:

```
<html>
  <head>
    <title>Arithmetical Operators</title>
  </head>
  <body>
    <?php
      $a = 10;
      $b = 20;
      /* If condition is true then assign a to result otheriwse b */
      $result = ($a > $b ) ? $a :$b;
           echo "TEST1 : Value of result is $result<br/>";
   /* If condition is true then assign a to result otheriwse b */
$result = ($a < $b ) ? $a :$b;
      echo "TEST2 : Value of result is $result<br/>";
    ?>
  </body>

</html>
```

This will produce the following

result: TEST1 : Value of result is 20

TEST2 : Value of result is 10

The ternary operator allows us to simplify some PHP conditional statements. We'll see how it can be used, with test-driven development and refactoring, to simplify code like:

```
<?php

$result =
null; if
(5>3) {
   $result = "Bigger";
} else {
   $result = "Less";
}
```

Written using a ternary operator, we can write the above comparison as:

```
<?php
$result = 5 > 3 ? "Bigger" : "Less";
```

This is obviously a much simpler way to write relatively easy to understand conditional statements and something we should consider when writing future code,

Expressions

An expression is a bit of PHP that can be evaluated to produce a value. The simplest expressions are literal values and variables. A literal value evaluates to itself, while a variable evaluates to the value stored in the variable. More complex expressions can be formed using simple expressions and operators.

An operator takes some values (the operands) and does something (for instance, adds them together). Operators are written as punctuation symbols—for instance, the + and - familiar to us from math. Some operators modify their operands, while most do not.

- Topic 2: User defined data types

- **Arrays**

Arrays in PHP is a type of data structure that allows us to store multiple elements of similar data type under a single variable thereby saving us the effort of creating a different variable for every data. The arrays are helpful to create a list of elements of similar types, which can be accessed using their index or key. Suppose we want to store five names and print them accordingly. This can be easily done by the use of five different string variables. But if instead of five, the number rises to hundred, then it would be really difficult for the user or developer to create so much different variables. Here array comes into play and helps us to store every element within a single variable and also allows easy access using an index or a key.

An array is created using an **array()** function in PHP. There are basically three types of arrays in PHP:

Indexed or Numeric Arrays:

An array with a numeric index where values are stored linearly.

```
<?php

// One way to create an indexed array
$name_one = array("Zack", "Anthony", "Ram", "Salim", "Raghav");
// Accessing the elements directly
echo "Accessing the 1st array elements directly:\n"; echo $name_one[2], "\n"; echo $name_one[0],
"\n"; echo $name_one[4], "\n";
// Second way to create an indexed array
$name_two[0] = "ZACK";
$name_two[1] = "ANTHONY";
$name_two[2] = "RAM";
$name_two[3] = "SALIM";
$name_two[4] = "RAGHAV";
// Accessing the elements directly echo "Accessing the 2nd array elements directly:\n";  echo
$name_two[2], "\n"; echo $name_two[0],
"\n"; echo $name_two[4], "\n";

?>
```

- **Associative Arrays:** An array with a string index where instead of linear storage, each value can be assigned a specific key.

- Example:
- filter_none
- edit
- play_arrow
- brightness_4

```
<?php
// One way to create an associative array
$name_one = array("Zack"=>"Zara", "Anthony"=>"Any", "Ram"=>"Rani", "Salim"=>"Sara",
"Raghav"=>"Ravina");
// Second way to create an associative array
$name_two["zack"] = "zara";
$name_two["anthony"] = "any";
$name_two["ram"] = "rani";
$name_two["salim"] = "sara";
$name_two["raghav"] = "ravina";
// Accessing the elements directly
echo "Accessing the elements
directly:\n"; echo
$name_two["zack"], "\n";  echo
$name_two["salim"], "\n";  echo
$name_two["anthony"], "\n"; echo
$name_one["Ram"], "\n";
echo $name_one["Raghav"], "\n";
?>
```

Multidimensional Arrays: An array which contains single or multiple array within it and can be accessed via multiple indices.

```
<?php
// Defining a multidimensional array
$favorites =
   array( array(

      "name" => "Dave
      Punk", "mob" =>
      "5689741523",
      "email" => "davepunk@gmail.com",

   ),
      array(
      "name" => "Monty
      Smith", "mob" =>
      "2584369721",
      "email" => "montysmith@gmail.com",
   ),
      array(
      "name" => "John
      Flinch", "mob" =>
      "9875147536",
      "email" => "johnflinch@gmail.com",
   )
);
// Accessing elements echo "Dave Punk email-id is: " .
$favorites[0]["email"], "\n"; echo "John Flinch mobile number is: " . $favorites[1]["mob"];
?>
```

LO 1.3 – Introduce PHP comments

- Topic 1: Introduction to PHP comments

- **Purpose of comments**

The **PHP comment** syntax always begins with a special character sequence and all text that appears between the start of the **comment** and the end will be ignored.

The main **purpose** is to serve as a note to you, the web developer or to others who may view your website's source code

- **Types of comments comment in HTML**

In case you forgot what an HTML comment looked like, see our example below.

HTML Code:

```
<!-- This is an HTML Comment -->
```

Single Line Comment

While there is only one type of comment in HTML, PHP has two types. The first type we will discuss is the single line comment. The single line comment tells the interpreter to ignore everything that occurs on that line to the right of the comment. To do a single line comment type "//" or "#" and all text to the right will be ignored by PHP interpreter.

PHP Code:

```
<?php
echo "Hello World!"; // This will print out Hello World!
echo "<br />Psst...You can't see my PHP comments!"; // echo "nothing";
// echo "My name is
Humperdinkle!"; # echo "I don't do
anything either";
```

Display:

```
Hello World!
Psst...You can't see my PHP comments!
```

Notice that a couple of our echo statements were not evaluated because we commented them out with the single line comment. This type of line commenting is often used for quick notes about complex and confusing code or to temporarily remove a line of PHP code.

Multiple Line Comment

Similiar to the HTML comment, the multi-line PHP comment can be used to comment out large blocks of code or writing multiple line comments. The multiple line PHP comment begins with " /* " and ends with " */ ".

PHP Code:

```
<?php
/* This Echo statement will print out my message to
the the place in which I reside on. In other words, the
World. */ echo "Hello World!";
/* echo "My name is Humperdinkle!"; echo
"No way! My name is Uber PHP
Programmer!"; */
?>
```

Display:

```
Hello World!
```

- Topic 2: Use of PHP comments

Comments are usually written within the block of *PHP* code to explain the functionality of the code. Single line *comment* used for quick notes about complex code or to temporarily disable a line of *PHP* code. You need to add // or # before the code. multi-line *comment* used to *comment* out large blocks of code or writing multiple line *comments*. You need to add /* before and */ after the code.

Learning Unit 2 – Implement PHP logic

LO 2.1 – Use PHP control structures

- Topic 1: Application of program flow control
  - **Conditional statements**

Very often when you write code, you want tmo perform different actions for different conditions. You can use conditional statements in your code to do this. In PHP we have the following conditional statements:

- **If** statement - executes some code if one condition is true
- **If**...**else** statement - executes some code if a condition is true and another code if that condition is false
- **If**...**elseif** **...**else statement - executes different codes for more than two conditions
- Switch statement - selects one of many blocks of code to be executed

PHP - The if Statement

The if statement executes some code if one condition is true.

Syntax

if (*condition*) {

code to be executed if condition is true;

}

The example below will output "Have a good day!" if the current time (HOUR) is less than 20:

Example

```
<?php
$t = date("H");
if ($t < "20") {
    echo "Have a good day!";
}
?>
```

PHP - The if...else Statement

The **if. else** statement executes some code if a condition is true and another code if that condition is false.

Syntax

```
if (condition) {
    code to be executed if condition is true;
} else {
    code to be executed if condition is false;
}
```

The example below will output "Have a good day!" if the current time is less than 20, and "Have a good night!" otherwise:

Example

```
<?php
$t = date("H");
if ($t < "20") {
    echo "Have a good day!";
} else {
    echo "Have a good night!";
}
?>
```

PHP - The if...elseif....else Statement

The **if....elseif else** statement executes different codes for more than two conditions.

Syntax

```
if (condition) {
    code to be executed if this condition is true;
} elseif (condition) {
    code to be executed if this condition is true;
} else {
    code to be executed if all conditions are false;
}
```

The example below will output "Have a good morning!" if the current time is less than 10, and "Have a good day!" if the current time is less than 20. Otherwise it will output "Have a good night!":

Example

```
<?php
```

```
$t = date("H");
if ($t < "10") {
   echo "Have a good morning!";
} elseif ($t <
"20") { echo
"Have a good
day!";
} else {
   echo "Have a good night!";
}
?>
```

The PHP switch Statement

The switch statement is used to perform different actions based on different conditions.

Use the **switch** statement to **select one of many blocks of code to be executed**.

Syntax

```
switch (n) {
case label1: code to be executed if n=label1;
break;
case label2:     code to be executed if n=label2;
 break;
case label3:     code to be executed if n=label3;
break;
 ...
defa
 ult:
     code to be executed if n is different from all labels;
}
```

This is how it works: First we have a single expression *n* (most often a variable), that is evaluated once. The value of the expression is then compared with the values for each case in the structure. If there is a match, the block of code associated with that case is executed. Use break to prevent the code from running into the next case automatically. The default statement is used if no match is found.

```
$favcolor = "red";
```

Example

```
<?php
switch ($favcolor) {
case "red": echo "Your favorite color is red!";
break;
case "blue":echo "Your favorite color is blue!";
break;
case "green": echo "Your favorite color is green!";
break;
default:
echo "Your favorite color is neither red, blue, nor
green!";
}
?>
```

- **Iterative statements (PHP Loops)**

Often when you write code, you want the same block of code to run over and over again in a row. Instead of adding several almost equal code-lines in a script, we can use loops to perform a task like this.

In PHP, we have the following looping statements:

- **while** - loops through a block of code as long as the specified condition is true
- **do...while** - loops through a block of code once, and then repeats the loop as long as the specified condition is true
- **for - loops** through a block of code a specified number of times
- **foreach** - loops through a block of code for each element in an array

The PHP while Loop

The while loop executes a block of code as long as the specified condition is true.

Syntax

```
while (condition is true) {
  code to be executed;
}
```

The example below first sets a variable $x to 1 ($x = 1). Then, the while loop will continue to run as long as $x is less than, or equal to 5 ($x <= 5). $x will increase by 1 each time the loop runs ($x++):

$x = 1;

Example

```
<?php

while($x <= 5) {
   echo "The number is: $x <br>";
   $x++;
} ?>
```

PHP do...while Loop

The do...while loop will always execute the block of code once, it will then check the condition, and repeat the loop while the specified condition is true.

Syntax do {
   code to be executed;

} while (*condition is true*);

The example below first sets a variable $x to 1 ($x = 1). Then, the do while loop will write some output, and then increment the variable $x with 1. Then the condition is checked (is $x less than, or equal to 5?), and the loop will continue to run as long as $x is less than, or equal to 5:

Example

```
<?php
$x = 1;
do {
   echo "The number is: $x <br>";
   $x++;
} while ($x <= 5);
?>
```

Notice that in a do while loop the condition is tested AFTER executing the statements within the loop. This means that the do while loop would execute its statements at least once, even if the condition is false the first time.

The example below sets the $x variable to 6, then it runs the loop, **and then the condition is checked**:

Example

```
<?php
$x = 6;
```

Example

```
<?php

do {
    echo "The number is: $x <br>";
    $x++;
} while ($x <= 5);
?>
```

The PHP for Loop

The for loop is used when you know in advance how many times the script should run.

Syntax

for (*init counter; test counter; increment counter*) {

code to be executed;

}

Parameters:

- *init counter*: Initialize the loop counter value
- *test counter*: Evaluated for each loop iteration. If it evaluates to TRUE, the loop continues. If it evaluates to FALSE, the loop ends.
- *increment counter*: Increases the loop counter value The example below displays the numbers from 0 to 10:

Example

```
<?php
for ($x = 0; $x <= 10; $x++) {
    echo "The number is: $x
    <br>";
}
?>
```

The PHP foreach Loop

The ***for each*** loop works only on arrays, and is used to loop through each key/value pair in an array.

Syntax

foreach ($*array* as $*value*) {

    code to be executed;

}

For every loop iteration, the value of the current array element is assigned to $value and the array pointer is moved by one, until it reaches the last array element.

The following example demonstrates a loop that will output the values of the given array ($colors): ***Example***

```
<?php
$colors = array("red", "green", "blue", "yellow");
foreach ($colors as
   $value) { echo "$value
   <br>";
}
?>
```

LO 2.2 – Use PHP Functions

- **Topic 1: Application of PHP functions.**

Built-in functions

Functions are reusable bits of code that you use throughout a project. They help to better organize your application as well as eliminate the need to copy/paste repetitive pieces of code. In an ideal world an application should not have multiple functions doing the same thing.

PHP has a lot! of built in functions and while you are not expected to learn all of them at once there are some useful functions that can help in everyday programming and we will start from there.

String Manipulation Functions

Some of the most useful PHP functions are string manipulation functions. As the name suggests they manipulate strings.

Finding the length of a String

The **strlen()** functions works by passing a **string** or **variable** and then returns the total number of characters including spaces.

```
<?php
   $name = "Matthew ";

echo strlen($name); // 8
?>
```

Return part of a String

The **substr()** function is used to return a substring or part of a string. This function has 3 parameters which you can pass along. **Syntax** substr($string, $start,$length);

- $string – a string of text or a variable containing a string of text. Input must be at least one character.
- $start – think of the string as an array starting from [0]. If you wanted to start from the first character you would enter 0. A negative value will go to the end of the string.
- $length – (Optional) is the number of characters returned after the start character. If this value is less than or equal to the start value then it will return false.

```
<?php
   $name = "Matthew ";
   echo substr($name, 0, 5); // Matth   echo
substr($name, 2); // tthew   echo
substr($name, -6, 5); // tthew
?>
```

Note: If you don't provide a $length parameter, this function will just return the remainder of the string starting from the $start parameter.

Converting strings to UPPER or lower case

Two useful string functions that are simple to use are **strtoupper()** and **strtolower(),** these functions can convert your strings to all UPPERCASE or all lowercase.

They are very useful for case sensitive operations where you may require all characters to be lowercase for example.

```
<?php
   $name = "Matthew ";
   echo strtoupper($name); // MATTHEW

echo strtolower($name); // matthew
?>
```

Searching for a needle in a haystack!

Sometimes we need to find a substring within a string and to do that we can use **strpos().**
Syntax
strpos ($haystack,$needle,$offset)

- $haystack – this is the string in which you are going to find the $needle starting from [0].

- $needle – this is what you are going to search for in the $haystack.
- $offset – (Optional) search will start from this number of characters counted from the beginning of the string. Cannot be negative.

```
<?php
  $name = "Matthew ";    echo strpos($name, "M"); // 0
  echo strpos($name, "hew"); // 4
  echo strpos($name, "m"); // false ?>
```

Notice that the last example is false. That is because this function is case sensitive and could not find a match. We can almost make use of an if statement and some variables to make the strpos function more useful and meaningful.

```
<?php

  $string = "I am learning how to use PHP string functions!";

  $search = "JavaScript";
  if(strpos($string, $search) === false) {
     echo "Sorry we could not find '$search' in '$string'.";    }
?>
```

This would echo *"Sorry we could not find 'JavaScript' in 'I am learning how to use PHP string functions!'"*.

Arithmetic Manipulation Functions

As well as string manipulation function, PHP also has functions to manipulate numbers.

Rounding numbers.

One of the most commonly used math function is **round().** This function rounds numbers with decimal points up or down. You can round a number to an integer (whole number) or to a floating point (decimal numbers).

Syntax

round($val, $precision, $mode)

- $val – is the value to be rounded.
- $precision – (optional) number of decimal places to round to.
- $mode – the type of rounding that occurs and can be one of the following

```
<?php
```

```
   $number = 3.55776232;     echo
      round($number) . "<br/>".   // 4
      round($number, 1) . "<br/>". // 3.6
      round($number, 3) . "<br/>"; //3.558
      ?>
```

Other math functions for rounding are **ceil()** and **floor()** . If you simply need to round a number to the nearest whole number then these functions are better suited to that purpose.

- **ceil()** – rounds fractions up.
- **floor()** – rounds fractions down.

```
<?php
   $number = 3.55776232;

      echo ceil($number) .
      "<br/>". // 4
floor($number) . "<br/>"; // 3 ?>
```

Both functions require a **value** and unlike **round(),** do not have any additional parameters. Generating Random Numbers

Another very common math function is **rand()** which returns a random number between two numbers.

Syntax

rand($min, $max)

- $min – (optional) sets the lowest value to be returned. Default is 0
- $max – (optional) sets the maximum value to be returned. Default returns getrandmax().

Note: getrandmax() on some windows machines will return 32767. You will need to specify a $max value in order to return a larger number.

```
<?php
   echo rand(). "\n"; //10884
   echo rand(). "\n"; // 621
   echo rand(2, 10); //2
?>
```

Note: End-of-line terminator (\n) is used in Unix and Unix-like systems (most servers) to move the carriage down to a new line. Simply put, it creates a new line in source code.

Array Functions

Array or **array()** is itself a function that stores multiple values in to a single variable. Aside from the array() function there are a number of other functions to manipulate arrays, here we will look at some of the most common ones.

Adding New Elements

Adding new elements to the end of an array can be achieved by calling the **array_push()** function.
Syntax
array_push($array, $value1, $value2)

- $array – the array in which you are adding new elements to.
- $value1 – (required) is the first value to push onto the end of the $array.
- $value2 – (optional) is the second value to push onto the end of the $array.

You can push as many values as you need.

```
<?php

   $games = array();
   $array = array_push($games, "Farcry 4");
   $array = array_push($games, "Fallout 4");
   $array = array_push($games, "Metal Gear");
   $array = array_push($games, "Witcher 3");
    echo $array; // returns 4
var_dump($games); ?>
```

However is better to list each element in a single call like this:

```
<?php
   $games = array(); // target array
   $array = array_push($games, "Farcry 4", "Fallout 4", "Metal Gear", "Witcher 3");
   echo $array; // returns 4 var_dump($games); ?>
```

Both methods result in the same outcome. If you **echo** or **print array_push()** it will return the number of items to be pushed in to the array.

If you **var_dump()** the target array you will see something like this. array(4) { [0]

```
=> string(8) "Farcry 4"
[1] => string(9) "Fallout 4"
[2] => string(10) "Metal Gear"
[3] =>string(9) "Witcher 3"
```

}

Sorting an Array

As well as adding items to an array we sometimes need to be able to sort them. PHP has a handy function called "funnily enough" **sort()** to do just that.

Syntax

Sort ($array, $sort_flags)

- $array – the array in which you wish to sort.
- $sort_flags – (optional) modifies the sorting behavior.

By default the sorting behavior will reorganize an array alphabetically or numerically.

```
<?php
   $games = array( "Farcry 4", "Metal Gear", "Fallout 4", "Witcher 3",        "Batman"); sort($games); //
      array to sort echo join(", ", $games); //output - Batman, Fallout 4, Farcry 4, Metal Gear, Witcher 3
?>
```

In order to echo or print out sorted arrays we can use a function called **join()** which is an alias of another function called **implode().**

Join(glue, array) or implode(glue, array) functions return a string from the elements of an array and both have the same **syntax**.

- glue – (optional) also known as a **separator** is what to put between the array elements.
- array – (required) is the array to join to a string.

If you need to sort and reverse the order of any array then you can use a function called **rsort().**

It works exactly the same way as sort() except the output is reversed.

```
<?php
   $games = array( "Farcry 4", "Metal Gear", "Fallout 4", "Witcher 3", "Batman"); rsort($games); //
      array to sort echo join(", ", $games);
   //output - Witcher 3, Metal Gear, Farcry 4, Fallout 4,
Batman ?>
```

Summarize

By now you should have a good overview of some common PHP functions for handling strings, integers and arrays.

Now for a simple exercise, the goal is:

1. Create an array and put 5 elements in it.
2. Sort and count the array.
3. Randomly select an element from the array
4. echo or print out the select item in upper or lower case.

```
<?php
   // create an array
   $new_games = array();
   $games = array_push($new_games, "Farcry 4", "Metal Gear", "Fallout 4", "Witcher 3","Batman");
   // count number of items
   $total_items = count($new_games);
   // Sort the array    sort($new_games);
   // Randomly select an item
   $selected = rand(0, $total_items);
   // echo or print in upper or lower case
   echo strtoupper($new_games[$selected]);
?>
```

- **User-defined functions**

A user defined function declaration starts with the word "function":

Syntax *function functionName() {*
code to be executed;
}
Note: A function name can start with a letter or underscore (not a number).

Function names are NOT case-sensitive.

Tip: Give the function a name that reflects what the function does!

In the example below, we create a function named "writeMsg()". The opening curly brace ({) indicates the beginning of the function code and the closing curly brace (}) indicates the end of the function. The function outputs "Hello world!". To call the function, just write its name:

Example

```
<?php
function writeMsg()
{echo "Hello world!";
} writeMsg(); // call the function ?>
```

- **PHP Function Arguments**

Information can be passed to functions through arguments. An argument is just like a variable. Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument ($fname). When the familyName() function is called, we also pass along a name (e.g. Jani), and the name is used inside the function, which outputs several different first names, but an equal last name:

Example

```
<?php
function familyName($fname) {      echo "$fname Refsnes.<br>";
} familyName("Jani"); familyName("Hege"); familyName("Stale"); familyName("Kai Jim");
familyName("Borge");
?>
```

The following example has a function with two arguments ($fname and $year):

Example

```
<?php
function familyName($fname, $year) {
echo "$fname Refsnes. Born in $year <br>";
} familyName("Hege", "1975");
familyName("Stale", "1978");
familyName("Kai Jim", "1983"); ?>
```

- **PHP Default Argument Value**

The following example shows how to use a default parameter. If we call the function setHeight() without arguments it takes the default value as argument:

Example

```
<?php
function setHeight($minheight = 50) { echo "The height is : $minheight <br>";
}
setHeight(350);
setHeight(); // will use the default value of 50 setHeight(135); setHeight(80); ?>
```

- **PHP Functions - Returning values**

To let a function, return a value, use the return statement:

Example

```
<?php
function sum($x, $y) { $z = $x + $y;
return $z;
}
echo "5 + 10 = " . sum(5, 10) . "<br>";
echo "7 + 13 = " . sum(7, 13) . "<br>";
echo "2 + 4 = ". sum(2, 4);
?>
```

- **String manipulation functions**

strtolower():	Converts a **string** to lowercase letters
strtoupper():	Converts a **string** to uppercase letters
strtr():	Translates certain characters in a **string**
substr():	Returns a part of a **string**
wordwrap():	Wraps a string to a given number of characters

LO 2.3 – Implement PHP File Processing

- Topic 1: File formats supported by PHP

- **Image formats**

PHP is not limited to creating just HTML output. It can also be used to create and manipulate image files in a variety of different image formats, including ***GIF, PNG, JPEG, WBMP, and XPM***. Even more conveniently, PHP can output image streams directly to a browser.

- **File functions in PHP**

  - **Is_file():** This function Checks whether a file is a regular file.
  - **file_exists():** Checks whether or not a file or directory exists
  - **Ifopen():**Opens a file or URL
  - **fwrite():** Writes to an open file.
  - **fclose():** Closes an open file.
  - **fgets()** : Returns a line from an open file.
  - **fgetss():** Returns a line, with HTML and PHP tags removed, from an open file.
  - **copy():**Copies a file.
  - **unlink():** Deletes a file.
  - **file_get_contents():**Reads a file into a string.

LO 2.4 – Handle Errors and Exceptions

- Topic 1: Description of program error types

  - **Syntax errors**: errors due to the fact that the syntax of the language is not respected.
  - **Logical errors:** errors due to the fact that the specification is not respected.
  - **Semantic errors**: errors due to an improper use of program statements.

From the point of view of when errors are detected, we distinguish:

- **Compile time errors**: syntax errors and static semantic errors indicated by the compiler.
- **Runtime errors**: dynamic semantic errors, and logical errors, that cannot be detected by the compiler (debugging).

Topic 2: Introduction to error functions

✓ **die(): die**( ) **function** is used to display a message and exit the script. It may be used to print alternate message . Instead of showing error it will show the user friendly message. **die**( ) **function** is only used to print string messages .value of variables cannot print with **die**( ) **function**.

Example

Print a message and terminate the current script:

```
<?php
$site = "https://www.w3schools.com/";
fopen($site,"r") or die("Unable to connect to $site");
?>
```

- **Custom error functions**

Creating a custom error handler is quite simple. We simply create a special function that can be called when an error occurs in PHP. This function must be able to handle a minimum of two parameters (error level and error message) but can accept up to five parameters (optionally: file, line-number, and the error context):

Syntax error_function(error_level,error_message, error_file,error_line,error_context)

Parameter	Description
error_level	Required. Specifies the error report level for the user-defined error. Must be a value number. See table below for possible error report levels
error_message	Required. Specifies the error message for the user-defined error
error_file	Optional. Specifies the filename in which the error occurred
error_line	Optional. Specifies the line number in which the error occurred
error_context	Optional. Specifies an array containing every variable, and their values, in use when the error occurred

Topic 3: Application of exception block

Try and catch keywords

The finally block may also be specified after or instead of **catch** blocks. Code within the finally block will always be executed after the **try and catch** blocks, regardless of whether an exception has been thrown, and before normal execution resumes.

Examples

Throwing an Exception

```
<?php function inverse($x) {     if (!$x) {          throw new Exception('Division by zero.');
}
return 1/$x;
} try {
    echo inverse(5) . "\n";   echo inverse(0) . "\n"; } catch (Exception $e) {
echo 'Caught exception: ', $e->getMessage(), "\n";
}
// Continue execution
echo "Hello World\n"; ?>
```

```
    <?php
function inverse($x) { if (!$x) {
    throw new Exception('Division by zero.');
    }
    return 1/$x;
    } try {
        echo inverse(5) . "\n"; } catch (Exception $e) {
    echo 'Caught exception: ', $e->getMessage(), "\n";
    } finally {
    echo "First finally.\n";
    } try {
```

```
    echo inverse(0) . "\n"; } catch (Exception $e) {
echo 'Caught exception: ', $e->getMessage(), "\n";
} finally {
echo "Second finally.\n";
}
// Continue execution

echo "Hello World\n"; ?>
```

The above example will output:

```
 0.2

First finally.

Caught exception: Division by zero.

Second finally.

Hello World
```

Learning Unit 3 –Perform PHP MySQL Database interactions

LO 3.1 – Connect and access MySQL Database

Introduction to web server

- Installation and configuration of XAMPP or WAMP

Description of XAMPP components

XAMPP stands for Cross-Platform (**X**), Apache (**A**), MariaDB (**M**), PHP (**P**) and Perl (**P**). Since **XAMPP** is simple, lightweight Apache distribution it is extremely easy for developers to create a local web server for testing and deployment purposes.

- **Apache:** Apache is the most **widely** used web server software. Developed and maintained by Apache Software Foundation, Apache is an **open** source software available for free. It runs on 67% of all webservers in the world. It is fast, reliable, and secure.
- MySQL: **MySQL** is a full-featured relational database management system (RDBMS) that competes with the likes of Oracle DB and Microsoft's SQL Server. **MySQL** is sponsored by the Swedish company **MySQL** AB, which is owned by Oracle Corp. **MySQL** is written in C and C++ and is compatible with all major operating systems.
- **PHP: Hypertext Preprocessor** (or simply **PHP**) is a general-purpose programming language originally designed for web development. It was originally created by Rasmus Lerdorf in 1994
- **Perl:** is a family of two high-level, general-purpose, interpreted, dynamic programming languages. "Perl" usually refers to Perl 5, but it may also refer to its redesigned "sister language", Perl 6.

  Though Perl is not officially an acronym, there are various backronyms in use, including "Practical Extraction and Reporting Language". Perl was originally developed by Larry Wall in 1987 as a general-purpose Unix scripting language to make report processing easier.

Test the web server

- Actions module for the Apache
- http://localhost or 127.0.0.1
- Splash screen

- Topic 1: Application of connection object and method

- **Connection object**

$servername = "localhost";

$username = "username";

$password = "password";

$dbname = "myDB";

- **Mysqli_connect**

Before we can access data in the MySQL database, we need to be able to connect to the server:

Example (MySQLi Procedural)

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
// Create connection
$conn = mysqli_connect($servername, $username, $password);
// Check
connection if
(!$conn) {
   die("Connection failed: " . mysqli_connect_error());
}
echo "Connected successfully";
?>
```

Example (MySQLi Object-Oriented)

```
<?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";
```

?>

- **PDO (PHP Data Objects)**

- Topic2: Application database access methods

- **Mysqli_select**

Definition and Usage. The **mysqli_select_db**() function is used to change the default database for the connection.

- **Mysqli_query**

The **mysqli_query**() function is used to simplify the act of performing a query against the database represented by the link parameter.

Example
Perform queries against the database:

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno())
  {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
  }
// Perform queries
mysqli_query($con,"SELECT * FROM Persons");
mysqli_query($con,"INSERT INTO Persons
(FirstName,LastName,Age) VALUES
('Glenn','Quagmire',33)"); mysqli_close($con); ?>
```

Note: To insert using form, values are replaced by values declared for receiving value from the form.

Mysqli_num_rows

The mysqli_num_rows() function returns the number of rows in a result set.

Syntax
mysqli_num_rows(*result*);

```
$rowcount=mysqli_num_rows($result);
```

Example

```
<?php
$link = mysqli_connect("localhost", "my_user", "my_password", "world");
/* check connection */ if
```

```
(mysqli_connect_errno()) {
   printf("Connect failed: %s\n", mysqli_connect_error());     exit();
}
if ($result = mysqli_query($link, "SELECT Code, Name FROM Country ORDER BY Name"))
{
   /* determine number of rows result set */
   $row_cnt = mysqli_num_rows($result);
   printf("Result set has %d rows.\n", $row_cnt);
   /* close result set */    mysqli_free_result($result);
}
/* close connection
*/
mysqli_close($link);
?>
```

Mysqli_fetch_array

The **mysqli_fetch_array()** function fetches a result row as an associative array, a numeric array, or both.

Note: Fieldnames returned from this function are case-sensitive.

mysql_fetch_assoc() is equivalent to calling **mysql_fetch_array**() with MYSQL_ASSOC for the optional second parameter and it doesn't really need to exist because it just returns the key names instead of Numeric keys and key names which happens in **mysql_fetch_array**.'Associative arrays' returned by **mysql_fetch_assoc**().

Syntax
mysqli_fetch_array(*result,resulttype*)*;*
Example:

```
<?php
$con=mysqli_connect("localhost","my_user","my_password","my_db");
// Check connection
if (mysqli_connect_errno())
  {
  echo "Failed to connect to MySQL: " . mysqli_connect_error();
  }
$sql="SELECT Lastname,Age FROM Persons ORDER BY Lastname";
$result=mysqli_query($con,$sql);
// Numeric array
$row=mysqli_fetch_array($result,MYSQLI_NUM); printf ("%s (%s)\n",$row[0],$row[1]);
```

```
// Associative array
$row=mysqli_fetch_array($result,MYSQLI_ASSO C); printf ("%s (%s)\n",$row["Lastname"],$row["Age"]);
// Free result set
mysqli_free_result($result)
;
mysqli_close($con);
?>
```

– Mysqli_close

Close a previously opened database connection:

```
mysqli_close($conn);
```

LO 3.2 – Implement database CRUD operations

The **Data** Access Layer communicates with the **Data** Storage Layer to perform **CRUD operations**. **CRUD** represents an acronym for the database operations **Create, Read, Update, and Delete**. The communication between two layers could be in the form of ad hoc **SQL** statements such as INSERT, SELECT, UPDATE, and DELETE.

- Topic 1: Manipulation of database table records

– Insert record

The INSERT INTO statement is used to add new records to a MySQL table:

```
<?php
$servername = "localhost";
$username = "username";
$password = "";
$dbname = "myDB";
// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection if
(!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
$sql = "INSERT INTO MyGuests (firstname, lastname, email)
VALUES ('John', 'Doe', 'john@example.com')";
if (mysqli_query($conn, $sql)) {       echo
"New record created successfully";
} else {
    echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}
mysqli_close($conn); ?>
```
Note: *To insert using form, values are replaced by values declared for receiving value from the form.*

Retrieve records

The SELECT statement is used to select data from one or more tables:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection if
(!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
$sql = "SELECT id, firstname, lastname FROM MyGuests";
$result = mysqli_query($conn, $sql);
if (mysqli_num_rows($result) > 0) {
    // output data of each row
    while($row = mysqli_fetch_assoc($result)) {
        echo "id: " . $row["id"]. " - Name: " . $row["firstname"]. " " . $row["lastname"]. "<br>";
    }
} else {
    echo "0 results";
}
mysqli_close($conn);
?>
```

Retrieve one record

```
$result = mysqli_query($conn, $sql);

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

Update record

The UPDATE statement is used to update existing records in a table:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection if
(!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
$sql = "UPDATE MyGuests SET lastname='Doe' WHERE id=2";

if (mysqli_query($conn, $sql)) { echo
"Record updated successfully";
} else {
   echo "Error updating record: " . mysqli_error($conn); }
mysqli_close($conn);
?>
```

Delete record

The DELETE statement is used to delete records from a table:

```
<?php
$servername = "localhost";
$username = "username";
$password = "password";
$dbname = "myDB";
// Create connection
$conn = mysqli_connect($servername, $username, $password, $dbname);
// Check connection if
(!$conn) {
    die("Connection failed: " . mysqli_connect_error());
}
// sql to delete a record
$sql = "DELETE FROM MyGuests WHERE id=3";
if (mysqli_query($conn, $sql)) { echo
"Record deleted successfully";
} else {
   echo "Error deleting record: " . mysqli_error($conn); }
mysqli_close($conn);
?>
```

Export and import records

Choose Excel File to Import Data

This HTML form with the file upload option is used to choose the excel source. On submitting this form, the excel file will be sent to the PHP to parse the data source. This file upload option will only allow the excel files to choose by using the *accept* attribute.

This code also contains the response HTML for displaying the message returned from PHP. This message is shown based on the type of response sent from PHP after excel import.

PHP Code to Import Excel Data to MySQL

Download and deploy PHP spreadsheet-reader library in your application *vendor* folder. Include the library path for accessing SpreadSheet-reader functions to read excel data into an array.

```
<?php
$conn = mysqli_connect("localhost","root","test","phpsamples");  require_once('vendor/php-
excel- reader/excel_reader2.php'); require_once('vendor/SpreadsheetReader.php');
if (isset($_POST["import"]))
{
  $allowedFileType = ['application/vnd.ms-
excel','text/xls','text/xlsx','application/vnd.openxmlformatsofficedocument.spreadsheetml.she
et'];
  if(in_array($_FILES["file"]["type"],$allowedFileType)){
        $targetPath = 'uploads/'.$_FILES['file']['name'];
        move_uploaded_file($_FILES['file']['tmp_name'], $targetPath);
        $Reader = new SpreadsheetReader($targetPath);
        $sheetCount = count($Reader->sheets()); for($i=0;$i<$sheetCount;$i++)
        {
           $Reader->ChangeSheet($i);
        foreach ($Reader as $Row)
        {
           $name = "";               if(isset($Row[0])) {
           $name = mysqli_real_escape_string($conn,$Row[0]);
           }
           $description = "";               if(isset($Row[1])) {
              $description = mysqli_real_escape_string($conn,$Row[1]);
           }
           if (!empty($name) || !empty($description)) {                $query = "insert into
           tbl_info(name,description) values('".$name."','".$description."')";
              $result = mysqli_query($conn, $query);
        "success";
        }
                 if (! empty($result)) {  $type = $message = "Excel Data Imported into the Database";
        } else {
        $type = "error";
        $message = "Problem in Importing Excel Data";
      }
      } }
else  {
      $type = "error";
      $message = "Invalid File Type. Upload Excel File.";
  }
} ?>
```

LO 3.3 – Manage Sessions and Cookies

- Topic 1: Introduction and use of cookies

Importance of cookies

A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With PHP, you can both create and retrieve cookie values.

```
<h2>Import Excel File into MySQL Database using PHP</h2>
   <div class="outer-container">        <form action="" method="post"
        name="frmExcelImport" id="frmExcelImport" enctype="multipart/formdata">
        <div>
           <label>Choose Excel
```

```
            <button type="submit" id="submit" name="import" class="btn- submit">Import</button>
        </div>
    </form>
  </div>
  <div id="response" class="<?php if(!empty($type)) { echo $type . " displayblock"; } ?>"><?php
if(!empty($message)) { echo $message; } ?></div>
<?php
  $sqlSelect = "SELECT * FROM tbl_info";
  $result = mysqli_query($conn, $sqlSelect);
if (mysqli_num_rows($result) > 0)
{
?>
  <table class='tutorial-table'>
    <thead>
      <tr>
        <th>Name</th>
        <th>Description</th>
      </tr>
    </thead>
<?php
  while ($row = mysqli_fetch_array($result)) {
?>
    <tbody>
    <tr>
      <td><?php echo $row['name']; ?></td>
      <td><?php echo $row['description']; ?></td>
    </tr>
<?php
  }
?>
    </tbody>
  </table>
<?php
}
?>
```

Types of Cookies:

Session based cookie which expire at the end of session and **Persistent cookies** which are written on hard disk.

Cookies life time

The distribution of lifetimes of 3rd party **cookies** is bimodal. Either they live less than a day, or they are likely to live for more than two weeks. **Cookies** seen by a campaign (any campaign!) live for longer.

Servlet API (Application programming interface)

Programming a PHP/Java Bridge servlet

You need to use **Servlet** API to create **servlets**. There are two packages that you must remember while using API, the javax.**servlet** package that contains the classes to support generic **servlet** (protocol- independent **servlet**) and the javax.**servlet**.http package that contains classes to support http **servlet**.

This shows how to create a servlet and how to call it from PHP code. Create a servlet using the Eclipse IDE

- Start Eclipse Java EE IDE for Web Developers
- Click on File/New/Dynamic Web Project
- Enter Project name: "MyWebApp", Target runtime: "Apache Tomcat v6.0", Configuration: "Default configuration for Apache Tomcat" Click on Finish
- Open the tree: Project Explorer/MyWebApp/WebContent/WEB-INF so that the *lib* directory becomes visible
- Drag and drop JavaBridge.jar and php-servlet.jar to the *lib* directory.
- Click on File/New/Servlet
- Enter Java package: "test", Class name: "Hello"
- Click Next, Click Next
- Uncheck "doPost", check "doGet" and "doPut"
- Click on Finish
- Add the following code to the doGet method body: response.getWriter().write("HelloServlet is running!");
- Add the following code to the doPut method body (press Control-Shift-o to automatically resolve references):

  RemoteHttpServletContextFactory ctx = new RemoteHttpServletContextFactory(this, getServletContext(), request, request, response); response.setHeader("X_JAVABRIDGE_CONTEXT", ctx.getId()); response.setHeader("Pragma", "no-cache");

  response.setHeader("Cache-Control", "no-cache");

  try { ctx.handleRequests(request.getInputStream(), response.getOutputStream());

```
} finally { ctx.destroy();
}
```

- Add the following method to the servlet: public String hello() {return "hello from MyServlet";}
- Click on the Run button. In the dialog box click on the Finish button. You should see the browser "http://localhost:8080/MyWebApp/Hello" with the message "HelloServlet is running!". Connect PHP with your servlet
- Drag and drop Java.inc to a directory and create the following PHP script "test.php":

```
<?php   define("JAVA_HOSTS",   "localhost:8080");
define("JAVA_SERVLET",      "/MyWebApp/Hello");
require_once("Java.inc");
  echo java_context()->getServlet()->hello();
?>
```

- Run the PHP script. For example with the command: php -n test.php It will invoke the hello() method and display its result.

Create cookies

A cookie is created with the **setcookie()** function.

Syntax

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

Only the *name* parameter is required. All other parameters are optional.

PHP Create/Retrieve a Cookie

The following example creates a cookie named "user" with the value "John Doe". The cookie will expire after 30 days (86400 * 30). The "/" means that the cookie is available in entire website (otherwise, select the directory you prefer).

We then retrieve the value of the cookie "user" (using the global variable $_COOKIE). We also use the **isset()** function to find out if the cookie is set:

Example

```
<?php
$cookie_name = "user";
$cookie_value = "John Doe";
```

```
setcookie($cookie_name, $cookie_value, time() + (86400 * 30), "/"); // 86400 =
1 day ?>
```

Retrieve cookies value

```
<html>
<body>
<?php
if(!isset($_COOKIE[$cookie_name])) {
    echo "Cookie named '" . $cookie_name . "' is not set!";
} else {
    echo "Cookie '" . $cookie_name . "' is set!<br>";
echo "Value is: " . $_COOKIE[$cookie_name];
}
?>
</body>
</html>
```

Delete cookies

To delete a cookie, use the ***setcookie()*** function with an expiration date in the past:

Example

```
<?php
// set the expiration date to one hour
ago setcookie("user", "", time() - 3600);
?>
<html>
<body>
<?php
echo "Cookie 'user' is deleted.";
?>
</body>
</html>
```

- Topic 2: Introduction and use of sessions

Importance of sessions

A session is a way to store information (in variables) to be used across multiple pages.

When you work with an application, you open it, do some changes, and then you close it. This is much like a Session. The computer knows who you are. It knows when you start the application and when you end. ***But on the internet there is one problem: the web server does not know who you are or what you do, because the HTTP address doesn't maintain state***.

Session variables solve this problem by storing user information to be used across multiple pages (e.g. username, favorite color, etc). By default, session variables last until the user closes the browser.

So; Session variables hold information about one single user, and are available to all pages in one application.

- **Session life time**

The session will last for **1440 seconds** (**24 minutes**). You're searching for gc_maxlifetime

- **Create sessions**

A session is started with the **session_start()** function.

Session variables are set with the PHP global variable: $_SESSION.

Now, let's create a new page called "demo_session1.php". In this page, we start a new PHP session and set some session variables:

Example

```
<?php
// Start the
session
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// Set session variables
$_SESSION["favcolor"] = "green";
$_SESSION["favanimal"] = "cat";
echo "Session variables are set.";
?>
</body>
```

```
</html>
```

Destroy session variables

To remove all global session variables and destroy the session, use **session_unset()** and session_destroy():

Example

```
<?php
session_start();
?>
<!DOCTYPE html>
<html>
<body>

<?php
// remove all session
variables session_unset();
// destroy the session
session_destroy();
?>

</body>
</html>
```

L O 3.4: Manage dynamic Forms

Topic 1: Application of HTML form processing methods

- **Embedded PHP scripts**

When you **embed PHP** code in an **HTML** file, you need to use the **.php** file extension for that file, so that your web server knows to send the file to **PHP** for processing.

- **POST method**

Information sent from a form with the POST method is **invisible to others** (all names/values are embedded within the body of the HTTP request) and has **no limits** on the amount of information to send. Moreover, POST supports advanced functionality such as support for multi-part binary input while uploading files to server.

However, because the variables are not displayed in the URL, it is not possible to bookmark the page.

$_POST is an array of variables passed to the current script via the HTTP POST method.

- **GET method**

Information sent from a form with the GET method is visible to everyone (all variable names and values are displayed in the URL). GET also has limits on the amount of information to send. The limitation is about 2000 characters. However, because the variables are displayed in the URL, it is possible to bookmark the page. This can be useful in some cases.

GET may be used for sending non-sensitive data.

Note: GET should NEVER be used for sending passwords or other sensitive information!

- **PHP form validation**

Create the form

Let's look at the form we used for the first tutorial and make a few updates to it.

```
<form action="php-form-processor.php" method="post">
Which is your favorite movie?
  <input type="text" name="formMovie" maxlength="50" value="<?=$varMovie;?>" />



  What is your name?
  <input type="text" name="formName" maxlength="50" value="<?=$varName;?>" />

  Please choose your gender?
  <select name="formGender">
    <option value="">Select...</option>
    <option value="M">Male</option>
    <option value="F">Female</option>
  </select>

    <input type="submit" name="formSubmit" value="Submit" />
    </form>
```

Getting the form data in the PHP script

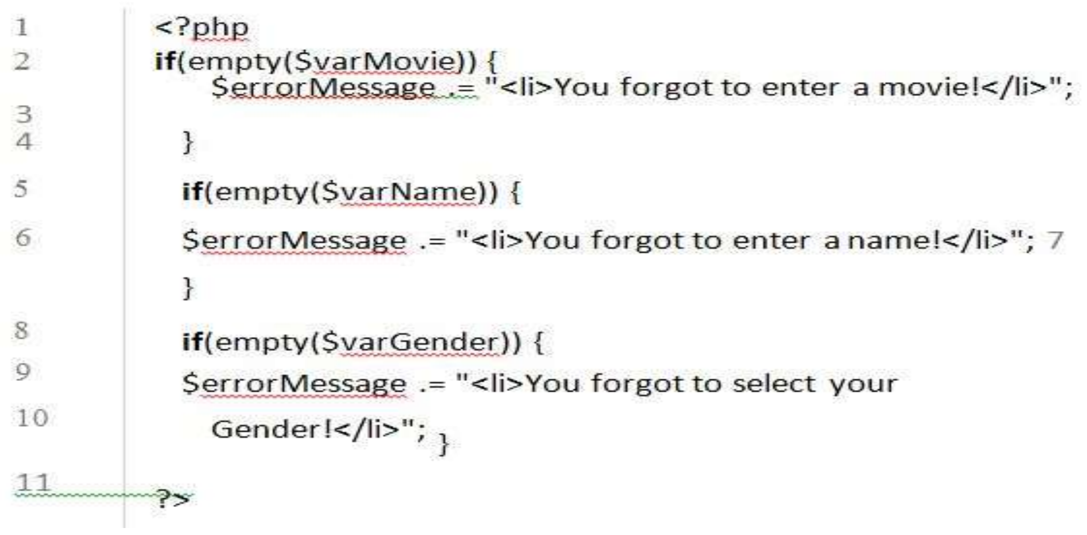
Let’s look at some PHP code to process this form.

```
<?php if($_POST['formSubmit'] ==
"Submit")
{

  $varMovie = $_POST['formMovie'];

  $varName = $_POST['formName'];

  $varGender = $_POST['formGender'];

  $errorMessage = "";


  // - - - snip - - -
}

?>
```

Select box input is accessed just like a text box. Now let’s put in some validation.

Validating the form data

It’s always a good idea to have a “blank” option as the first option in your select box. It forces the user to make a conscious selection from the box and avoids a situation where the user might skip over the box without meaning to. Of course, this requires validation.



```
<?php
if(empty($varMovie)) {
    $errorMessage .= "<li>You forgot to enter a movie!</li>";

  }
  if(empty($varName)) {
  $errorMessage .= "<li>You forgot to enter a name!</li>";
  }
  if(empty($varGender)) {
  $errorMessage .= "<li>You forgot to select your
    Gender!</li>"; }
?>
```

Learning Unit 4 – Use PHP Frameworks

LO 4.1 – Introduce Frameworks

A **PHP Framework** is a basic platform that allows us to develop web applications. In other words, it provides structure. By using a **PHP Framework**, you will end up saving loads of time, stopping the need to produce repetitive code, and you'll be able to build applications rapidly (RAD).

- Topic 1: Description of PHP frameworks.

CakePHP

CakePHP is a web development framework that uses the MVC model. It is a free open source framework for PHP that uses the Mode-View-Controller software design pattern. CakePHP reduces developmental costs and helps developers write less code.

CakePHP enables PHP users at all levels to rapidly develop robust web applications.

Extra Hot: Release: 1.3.0 beta & 1.2.6 Stable

Get it now!

CakePHP is a rapid development framework for PHP that provides an extensible architecture for developing, maintaining, and deploying applications. Using commonly known design patterns like MVC and ORM within the convention over configuration paradigm, CakePHP reduces development costs and helps developers write less code.

- download: 1.3-beta
- read the announcement
- view the changelog
- discover more hot features

Benefits of CakePHP:

- All segments work independently, which means, the developer may modify a segment without affecting others.

- Frequently updated with new features, making it feature-rich and highly secured.
- Due to its MVC pattern, creating powerful web applications is easy.
- Reduced development cost & time.
- Highly effective for scaffolding code generation.
- Easier to work on classes.
- Automated configuration for preferred settings.

✓ **Laravel**

Laravel is a free, open-source PHP web framework, created by Taylor Otwell and intended for the development of web applications following the model–view–controller architectural pattern and based on Symfony. Wikipedia

License: MIT License

Initial release: June 2011; 8 years ago

Stable release: 5.8.10 / April 4, 2019; 2 months ago

Written in: PHP

Developer(s): Taylor Otwell

✓ **Symfony**

Symfony is a PHP web application framework and a set of reusable PHP components/libraries. Symfony was published as free software on October 18, 2005 and released under the MIT license. Wikipedia

Original author(s): Fabien Potencier License: MIT license

Stable release: 4.2.7 / 2019-04-

17 Written in: PHP

Initial release: 22 October 2005

✓ **Zend**

Zend Framework is an open source, object-oriented web application framework implemented in PHP 7 and licensed under the New BSD License. The framework is basically a collection of professional PHP-based packages. Wikipedia

Developed by: Zend Technologies

Stable release: 3.0.0 / June 28, 2016; 2 years ago

Initial release date: 2005

License: New BSD license

Written in: PHP

- ✓ **Phalcon**

Phalcon is a PHP web framework based on the model–view–controller pattern. Originally released in 2012, it is an open-source framework licensed under the terms of the BSD License.

Stable release: 3.4.4 / 30 June 2019; 26 days ago Initial release date: November 14, 2012

Developer(s): Andres Gutierrez and others

License: BSD License

Platforms: Unix, Linux, macOS, Microsoft Windows

Written in: C, PHP

LO 4.2: Implement the CakePHP framework

- Topic 1: Installation of CakePHP 3.x

Before starting you should make sure your php version is up to date: php –v through CLI You should have 5.6.0 or higher. The cakephp is insatalled through Composer. CakePHP uses Composer, a dependency management tool, as the officially supported method for installation.

To install composer on windows, download the executable file and run it.

- Topic 2 CakePHP API

- ✓ **Class components**

Base class for an individual Component. Components provide reusable bits of controller logic that can be composed into a controller. Components also provide request life-cycle callbacks for injecting logic at specific points.

- ✓ **Controller**

Application controller class for organization of business logic. Provides basic functionality, such as rendering views inside layouts, automatic model availability, redirection, callbacks, and more.

- ✓ **Components collection**

A collection is an immutable list of elements with a handful of functions to iterate, group, transform and extract information from it.

- ✓ **EventManager**

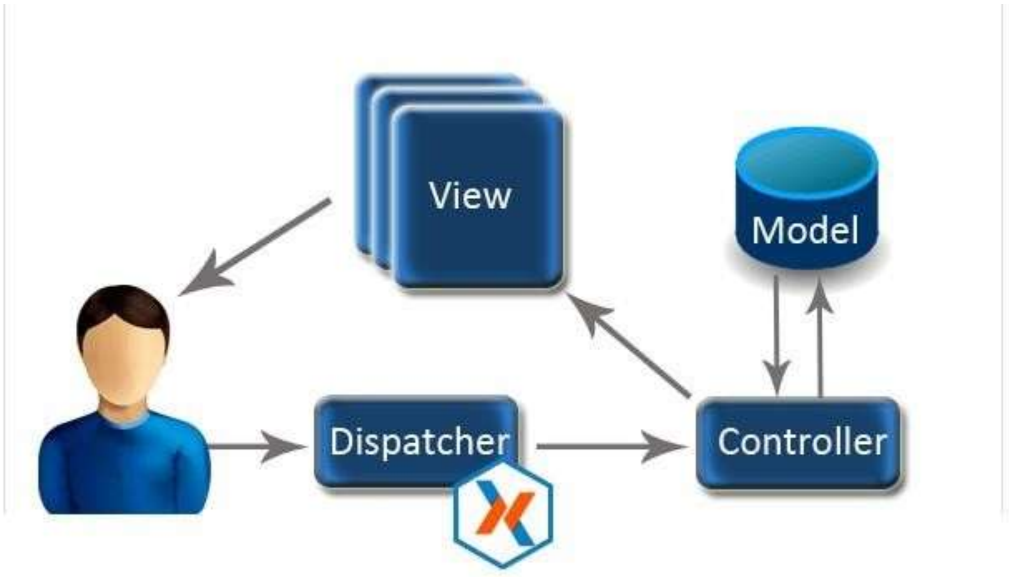
The event manager is responsible for keeping track of event listeners, passing the correct data to them, and firing them in the correct order, when associated events are triggered. You can create multiple instances of this object to manage local events or keep a single instance and pass it around to manage all events in your app.

- ✓ **EventListener**

An **event listener** is a procedure or function in a computer program that waits for an **event** to occur. The **listener** is programmed to react to an input or signal by calling the **event's** handler.

Topic 3: Develop a web application using CakePHP framework.

CakePHP MVC Architecture



Understand The MVC Architecture of CakePHP

The MVC patter has three components, namely Model, View and Controller, and each has its different role.

Model:

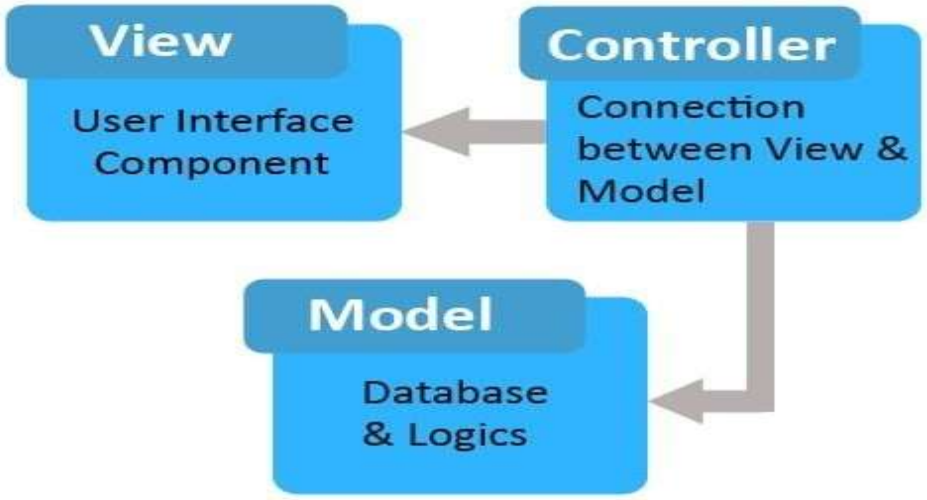
The Model Layer is one that represents the logical part of your application. It retrieves the data and convert it into meaningful concepts.

View:

As the name suggests, the View Layer shows the output to the end user. It processes the inputs provided by the Model Layer and generates specific output for the end user.

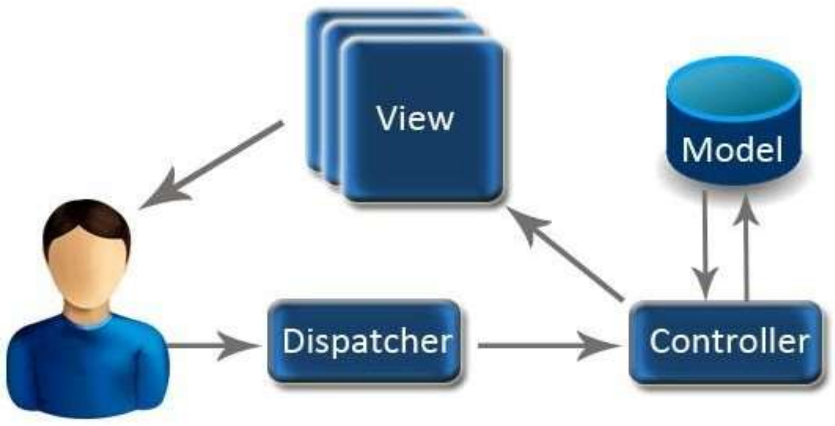
Controller:

The Controller Layer is responsible for handling all requests coming from the users. It collects the inputs from users and coordinate for the Model and View codes.



Model Layer is for database applications, View Layer is for graphical user interface implementation and Controller Layer is for business logics. Thus, these three layers combine together to make the entire process faster, easier and smoother. Also, each layer may work independently.

CakePHP Request Cycle:



The CakePHP Request Cycle starts with a user making a request for a page or resource on your application. The request first goes to a dispatcher that selects the appropriate controller object to process it. The controller then communicates with the Model Layer to process the data-fetching requirements. After that, the request is sent to the View Layer to generate the output for the user. After the output is generated, it is served to the user.

Naming conventions

By following conventions, you get free functionality, and you liberate yourself from the maintenance nightmare of tracking config files. Conventions also make for a very uniform development experience, allowing other developers to jump in and help.
Since it is a framework, CakePHP have its own naming convention standards to be used during PHP application development.

Whenever you have to create application using CakePHP you will need to create following files first.

- One File to define your model.
- One File to define your controller.
- One or More files to represent the OUTPUT called view files.

So, while creating above files you will have to follow CakePHP naming convention standards. And it’s very important to the PHP developer to follow these standards.

A. Database table and field naming convention

- Table name must be in lowercase & plural. If there are more than one word in the table name then words should be separated by underscore.
    - **Example** : users, consultant_services, etc.
- Field-Name should also be in lowercase and if more than one word then separated by underscore.
    - **Example** : id, consultant_service_name
- ForeignKey name should be singular and tablename_id

  O **Example** : users_id or consultant_services_id

B. Model naming Convention

- Model file names are upper camel case, more than one word should be separated by underscore.
    - **Example** : User.php or Consultant_service.php
- Model class name are singular, usually this should be the database table name but in singular format. o **Example** : User or ConsultantService

Note – Model Files are located at app/models folder

C. Controller naming Convention

- Controller file names are plural and upper camel case, and filenames also end with 'Controller.php'.
    - **Example** : UsersController.php or ConsultantservicesController.php

Note – Controller Files are located at app/Controller folder

View naming Convention View files are located at /app/views and into that a folder is created with its respective controller name, if more than one word then separated by underscore. View file names are function_name.ctp (The function name is the name of the function you have defined in your appropriate Controller file.)

Learning Outcome 4.3 : Implement the Laravel framework

- Topic 1: Installation of Laravel framework

How to install Laravel with XAMPP on Windows Using Composer

In this article we will give you a step-by-step guide on how to **install Laravel with XAMPP on Windows** Using Composer. Through Composer it's simple and easy to **install Laravel with XAMPP**.

# Server Requirements

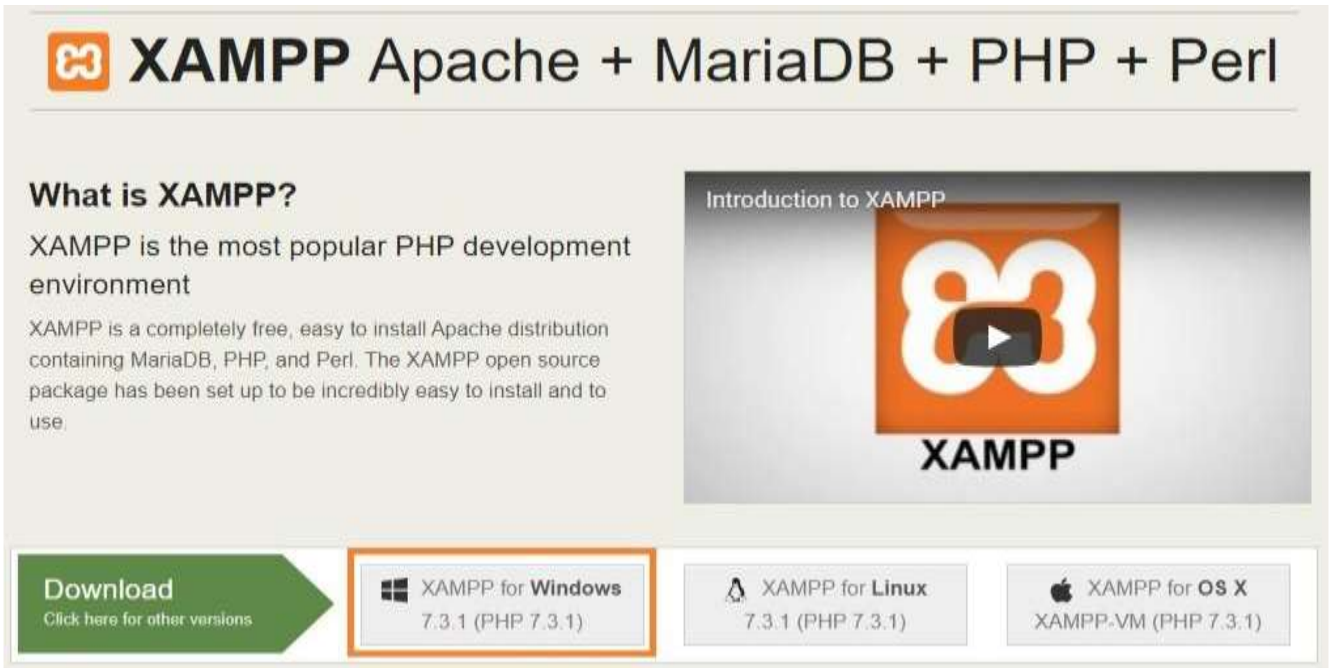
Since we want to work with the latest version of Laravel 5.6, make sure your server meets the following requirements:

- PHP >= 7.1.3
- OpenSSL PHP Extension
- PDO PHP Extension
- Mbstring PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension
- Ctype PHP Extension
- JSON PHP Extension

For more information, you can check the Server Requirements from Laravel official site.

# Install XAMPP

First thing you need to do is download the latest version of XAMPP that supports PHP 7.3.1 and Install it on Windows. For more details, see our step by step guide on How To Download And Install XAMPP On Windows.



XAMPP Apache + MariaDB + PHP + Perl

What is XAMPP?

XAMPP is the most popular PHP development environment

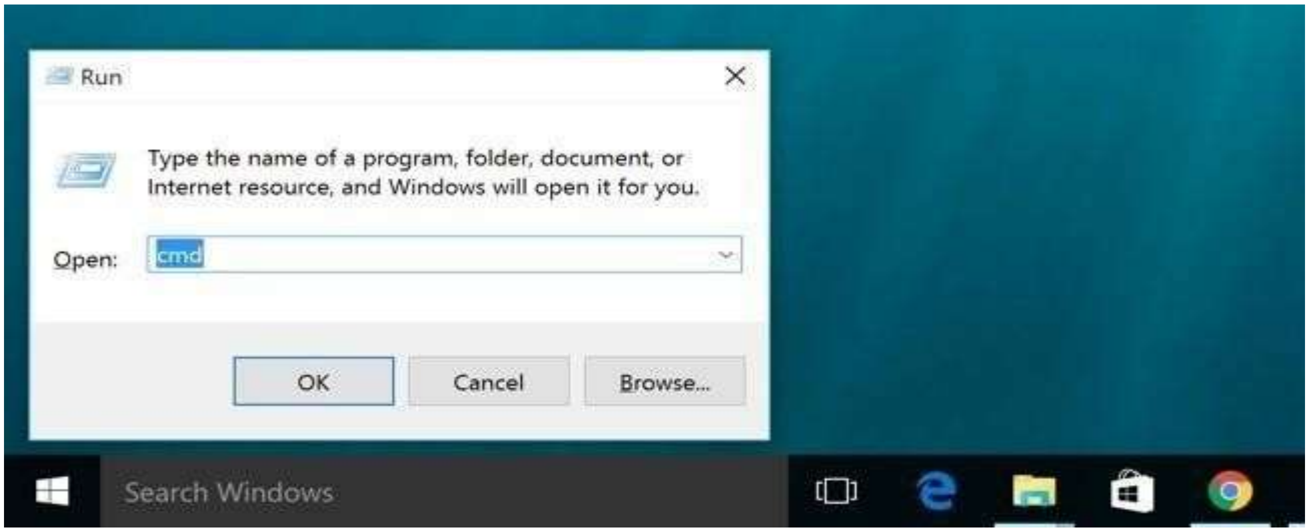
XAMPP is a completely free, easy to install Apache distribution containing MariaDB, PHP, and Perl. The XAMPP open source package has been set up to be incredibly easy to install and to use.

# Composer

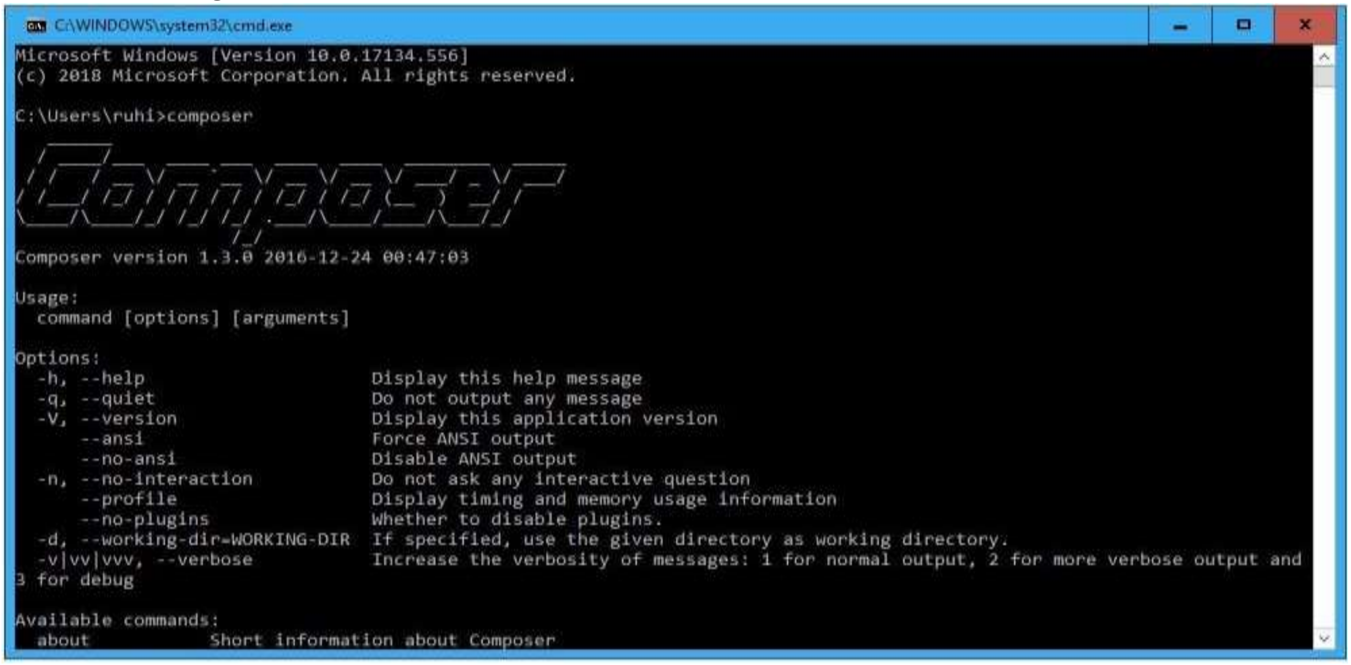
Once you have downloaded and installed XAMPP on Windows, then you need to do is Download Composer for Windows and Install it. For more details, see our step by step guide on How to Install Composer On Windows With XAMPP.

Laravel utilizes Composer to manage its dependencies. It must installed before setting up Laravel.

Once you successfully installed the Composer, open the command prompt. To open it, press **Win + R** keys on the keyboard, type in **cmd** and press the **OK** button


Run
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open: cmd
OK
Cancel
Browse...
Search Windows

Then type **composer** and press enter in the command prompt and you will get following response like in the below image.

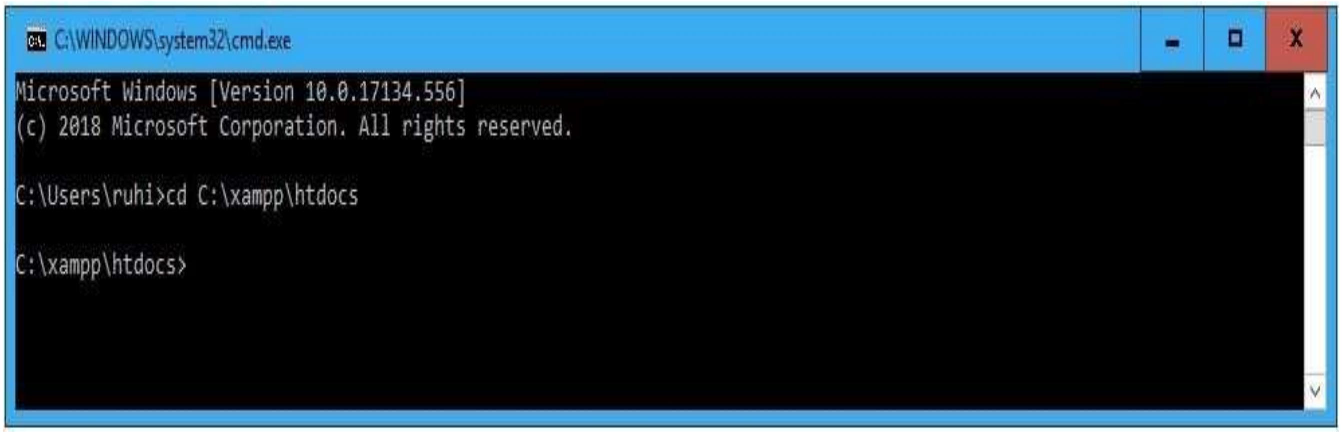


```
Microsoft Windows [Version 10.0.17134.556]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\ruhi>composer
   ______
  / ____/___  ____ ___  ____  ____  ________  _____
 / /   / __ \/ __ `__ \/ __ \/ __ \/ ___/ _ \/ ___/
/ /___/ /_/ / / / / / / /_/ / /_/ (__  )  __/ /
\____/\____/_/ /_/ /_/ .___/\____/____/\___/_/
                    /_/
Composer version 1.3.0 2016-12-24 00:47:03

Usage:
  command [options] [arguments]

Options:
  -h, --help                     Display this help message
  -q, --quiet                    Do not output any message
  -V, --version                  Display this application version
      --ansi                     Force ANSI output
      --no-ansi                  Disable ANSI output
  -n, --no-interaction           Do not ask any interactive question
      --profile                  Display timing and memory usage information
      --no-plugins               Whether to disable plugins.
  -d, --working-dir=WORKING-DIR  If specified, use the given directory as working directory.
  -v|vv|vvv, --verbose           Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and
3 for debug

Available commands:
  about               Short information about Composer
```

# Install Laravel Framework

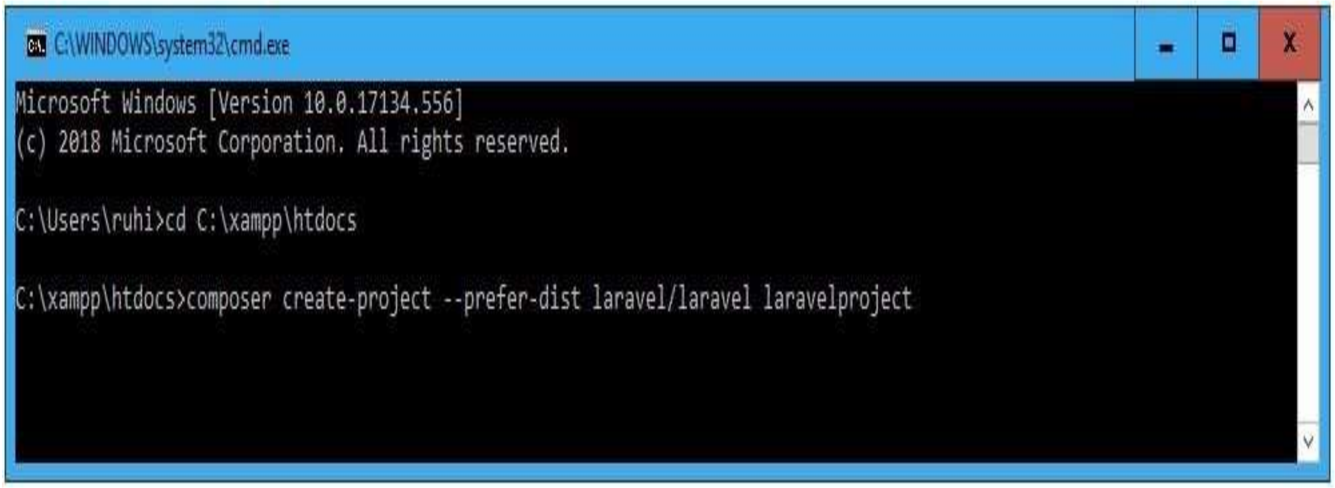
To install Laravel, First of all, you have to go **C:/xampp/htdocs** directory**,** to navigate it type following command in your command prompt:

cd c:/xampp/htdocs



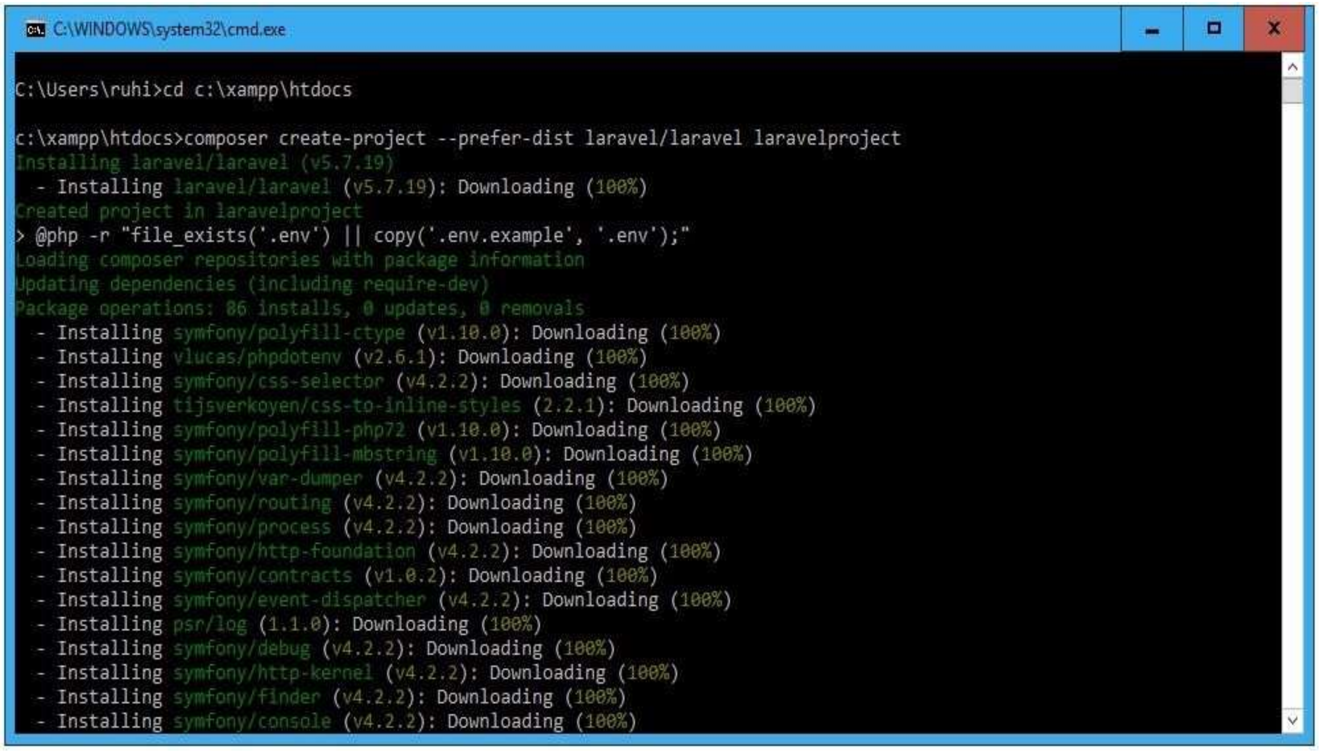
```
Microsoft Windows [Version 10.0.17134.556]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\ruhi>cd C:\xampp\htdocs

C:\xampp\htdocs>
```

Now we are ready to install Laravel, run the following command to install latest Laravel version: composer create-project --prefer-dist laravel/laravel laravelproject



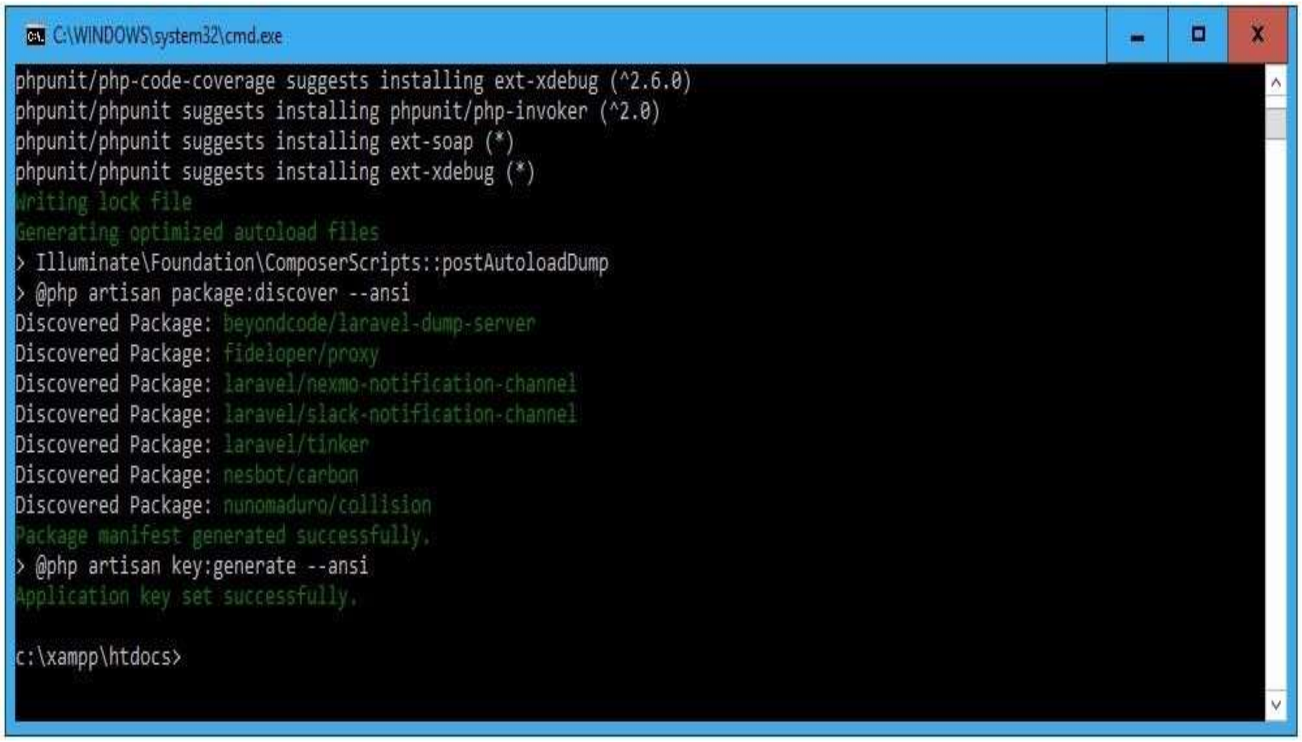
```
Microsoft Windows [Version 10.0.17134.556]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\ruhi>cd C:\xampp\htdocs

C:\xampp\htdocs>composer create-project --prefer-dist laravel/laravel laravelproject
```

After running this command it should start downloading dependencies that are required to create the Laravel project.



```
C:\Users\ruhi>cd c:\xampp\htdocs

c:\xampp\htdocs>composer create-project --prefer-dist laravel/laravel laravelproject
Installing laravel/laravel (v5.7.19)
  - Installing laravel/laravel (v5.7.19): Downloading (100%)
Created project in laravelproject
> @php -r "file_exists('.env') || copy('.env.example', '.env');"
Loading composer repositories with package information
Updating dependencies (including require-dev)
Package operations: 86 installs, 0 updates, 0 removals
  - Installing symfony/polyfill-ctype (v1.10.0): Downloading (100%)
  - Installing vlucas/phpdotenv (v2.6.1): Downloading (100%)
  - Installing symfony/css-selector (v4.2.2): Downloading (100%)
  - Installing tijsverkoyen/css-to-inline-styles (2.2.1): Downloading (100%)
  - Installing symfony/polyfill-php72 (v1.10.0): Downloading (100%)
  - Installing symfony/polyfill-mbstring (v1.10.0): Downloading (100%)
  - Installing symfony/var-dumper (v4.2.2): Downloading (100%)
  - Installing symfony/routing (v4.2.2): Downloading (100%)
  - Installing symfony/process (v4.2.2): Downloading (100%)
  - Installing symfony/http-foundation (v4.2.2): Downloading (100%)
  - Installing symfony/contracts (v1.0.2): Downloading (100%)
  - Installing symfony/event-dispatcher (v4.2.2): Downloading (100%)
  - Installing psr/log (1.1.0): Downloading (100%)
  - Installing symfony/debug (v4.2.2): Downloading (100%)
  - Installing symfony/http-kernel (v4.2.2): Downloading (100%)
  - Installing symfony/finder (v4.2.2): Downloading (100%)
  - Installing symfony/console (v4.2.2): Downloading (100%)
```

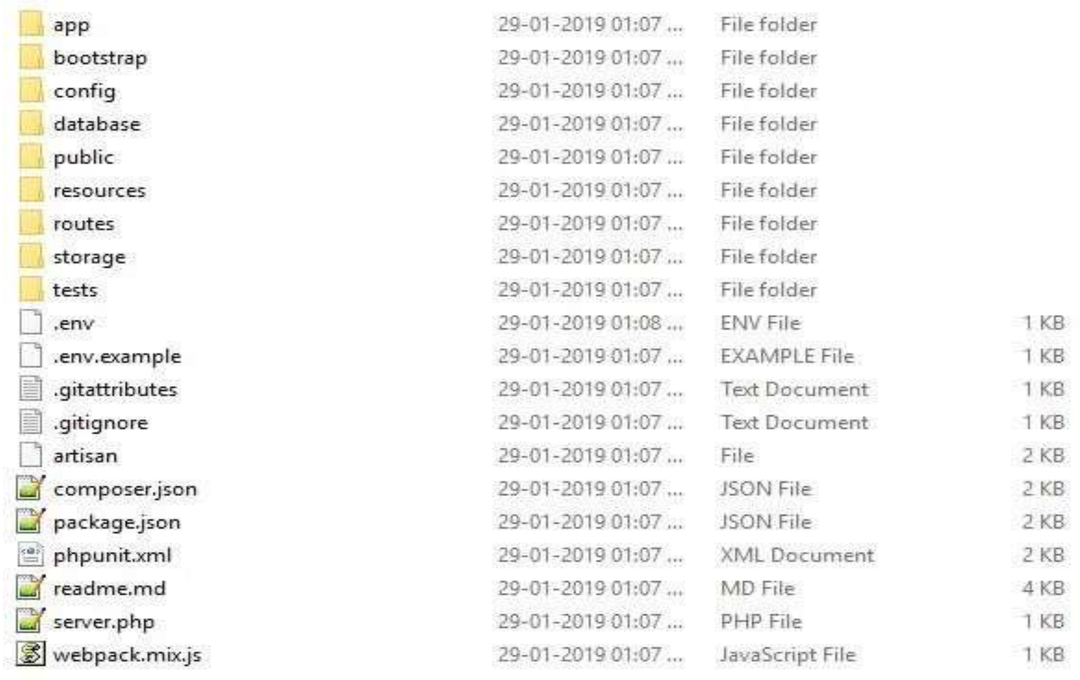
This installation may take a few minutes after executing the above command, so wait until you get success message like in the below image.



```
phpunit/php-code-coverage suggests installing ext-xdebug (^2.6.0)
phpunit/phpunit suggests installing phpunit/php-invoker (^2.0)
phpunit/phpunit suggests installing ext-soap (*)
phpunit/phpunit suggests installing ext-xdebug (*)
Writing lock file
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: beyondcode/laravel-dump-server
Discovered Package: fideloper/proxy
Discovered Package: laravel/nexmo-notification-channel
Discovered Package: laravel/slack-notification-channel
Discovered Package: laravel/tinker
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Package manifest generated successfully.
> @php artisan key:generate --ansi
Application key set successfully.

c:\xampp\htdocs>
```

After executing this command, it create a folder **'laravelproject'** under **C:/xampp/htdocs** directory with all it's dependency.

When it completed, it will create following directory schema:



app	29-01-2019 01:07 ...	File folder	
bootstrap	29-01-2019 01:07 ...	File folder	
config	29-01-2019 01:07 ...	File folder	
database	29-01-2019 01:07 ...	File folder	
public	29-01-2019 01:07 ...	File folder	
resources	29-01-2019 01:07 ...	File folder	
routes	29-01-2019 01:07 ...	File folder	
storage	29-01-2019 01:07 ...	File folder	
tests	29-01-2019 01:07 ...	File folder	
.env	29-01-2019 01:08 ...	ENV File	1 KB
.env.example	29-01-2019 01:07 ...	EXAMPLE File	1 KB
.gitattributes	29-01-2019 01:07 ...	Text Document	1 KB
.gitignore	29-01-2019 01:07 ...	Text Document	1 KB
artisan	29-01-2019 01:07 ...	File	2 KB
composer.json	29-01-2019 01:07 ...	JSON File	2 KB
package.json	29-01-2019 01:07 ...	JSON File	2 KB
phpunit.xml	29-01-2019 01:07 ...	XML Document	2 KB
readme.md	29-01-2019 01:07 ...	MD File	4 KB
server.php	29-01-2019 01:07 ...	PHP File	1 KB
webpack.mix.js	29-01-2019 01:07 ...	JavaScript File	1 KB

# XAMPP Virtual Host

We need to configure a Virtual Host in XAMPP for a Laravel project and in this example we want to configure the domain **localhost.laravelproject.com** for our project. To do so, edit **httpdvhosts.conf** file which is located within **C:\xampp\apache\conf\extra\httpd-vhosts.conf** Add following code snippet at the end of your file:

```
# VirtualHost for localhost.laravelproject.com
<VirtualHost *:80>
   DocumentRoot
   "C:/xampp/htdocs/laravelproject/public" ServerName
   localhost.laravelproject.com
   <Directory "C:/xampp/htdocs/laravelproject/">
     Options Indexes MultiViews FollowSymLinks
     AllowOverride All
     Require all granted
   </Directory>
```

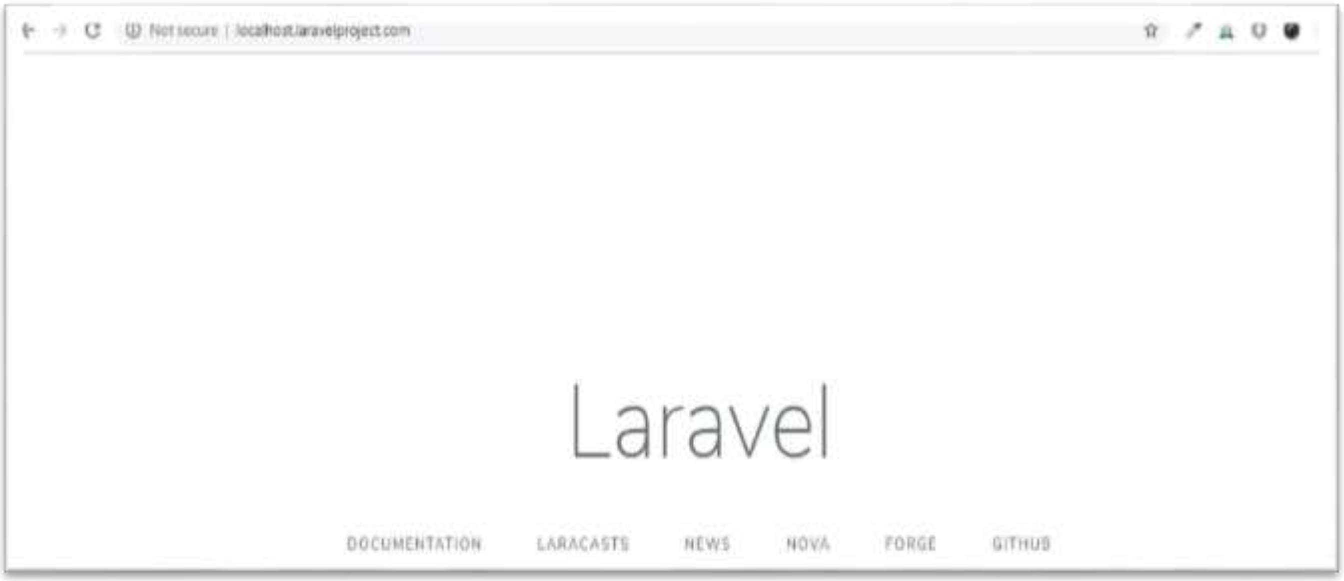
</VirtualHost>

After this, apache server is listening to **localhost.laravelproject.com** connections, but we have to configure our hosts file that allows to redirect **localhost.laravelproject.com** to the localhost, to do so, edit the **hosts** file which is located within **C:\Windows\System32\drivers\etc**

Add following code snippet at the end of your file:

```
127.0.0.1  localhost
127.0.0.1 127.0.0.1
127.0.0.1  localhost.laravelproject.com
```

Now everything is ready, restart the Apache server using the XAMPP control panel and type this URL **http://localhost.laravelproject.com** in your browser and press enter.



Topic 2: Laravel APIs

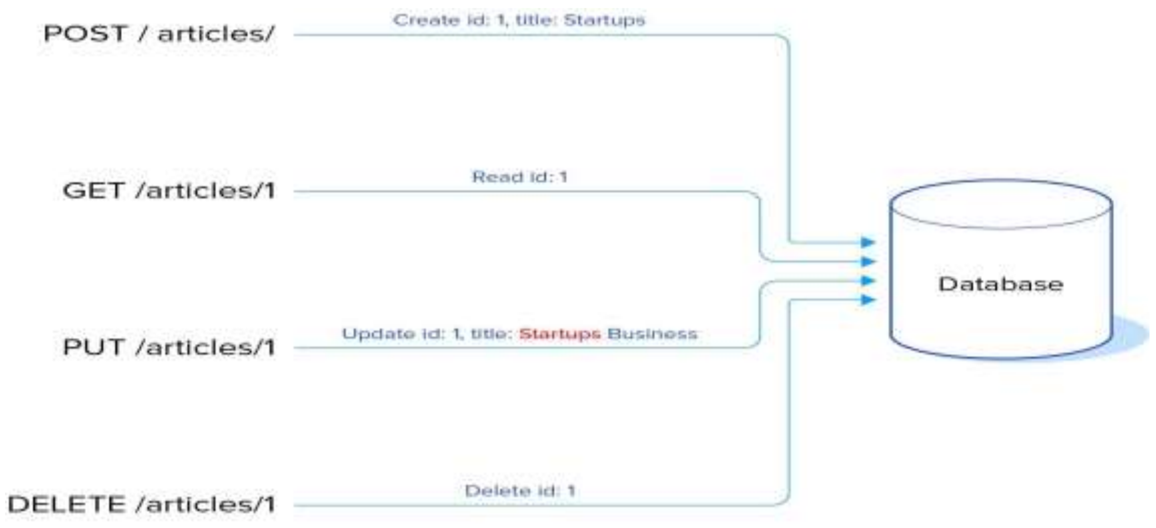
RESTful APIs

REST stands for **RE**presentational **S**tate **T**ransfer and is an architectural style for network communication between applications, which relies on a stateless protocol (usually HTTP) for interaction.

HTTP Verbs Represent Actions

In RESTful APIs, we use the HTTP verbs as actions, and the endpoints are the resources acted upon. We'll be using the HTTP verbs for their semantic meaning:

- GET: retrieve resources
- POST: create resources
- PUT: update resources
- DELETE: delete resources



Development of web application using Laravel

Step 1: Download Laravel and Other Project Files

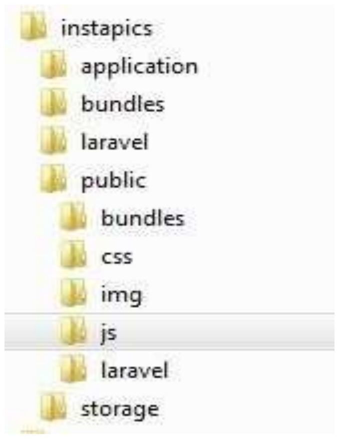
Before we begin, let's first ensure that we have a system that can support Laravel. According to the documentation, Laravel requires the following:

- **PHP 5.3.x** - Laravel makes use of a lot of PHP 5.3-specific features, like closures, late-static binding and namespaces.
- **The FileInfo library** - this is enabled by default in PHP 5.3, but on Windows systems, you might need to add the extension in your PHP.ini configuration file.
- **Mcrypt library** - this is used by Laravel for encryption and hash generation, and typically comes preinstalled with PHP.

Once we're done setting up the environment, let's download Laravel and all the libraries we'll be using for Instapics. Download the following files and place them within a web-accessible folder:

- **Laravel** - http://laravel.com (Currently v3.2.1)
- **Twitter Bootstrap** - http://twitter.github.com/bootstrap/ (Currently v2.0.4)
- **jQuery** - http://jquery.com (Currently v1.7.2)

Inside Laravel's root folder, you'll find a **public** folder - this is where all publicly accessible files should be stored. Laravel v3.2.1 has some premade folders inside the **public** folder for our assets, **css**, **img**, and **js** folders. Place the Twitter Bootstrap and jQuery files in their corresponding folders. At this point, your folder structure should look similar to the following:



```
instapics
  application
  bundles
  laravel
  public
    bundles
    css
    img
    js
    laravel
  storage
```

Twitter Bootstrap will have some files inside the css, img, and js folders, and jQuery will be inside the **js** folder.

Step 2: Setup Laravel's Encryption Key, Pretty URLs and Virtual Host

If you are new to Laravel, then you should know that you have the provision to create configuration files for Laravel application. After installing Laravel, you need to perform the permission writing for your **storage directory** along with the **bootstrap/cache**.

Next, you have to do is, generating the application key for session securing and encrypted data keys also. In case, the root directory doesn't have the **.env** file, in that case, you will have to rename the file **.env.example** to **.env** and run the command mentioned below where you've installed the Laravel: **php artisan key: generate**

You can see in the **.env** file the newly generated key. Moreover, it is also possible to configure time zone as well as a locale in the **config/app.php** file of your project.

Laravel allows us for running application for a diverse environment like testing, production, etc. For configuring your application's environment, you will need the **.env** file which is in the root directory of the project. When you install composer for Laravel, then this file gets generated or created automatically by the composer itself, but if you don't install then have to rename the specific file with the name **.env.example** to **.env** only.

You can configure the database for your application using the **config/database.php** file of your project. Setting the configuration constraint utilized by various databases can also be done, and moreover, Laravel also allowed us to use the default one as well.

Websites are regularly modified. As s a developer for this, you have to put your site in maintenance mode. In this advanced framework, it becomes easier to do that by using two artisan commands. Let's see how to use the commands:

For starting your project maintenance approach, the following command is required:

php artisan down

After changing the required stuff, when it is time to re-run your project, the following command is required:

php artisan up

Step 3: Setup Routing

Routing is one of the essential concepts in Laravel. Routing in Laravel allows you to route all your application requests to its appropriate controller. The main and primary routes in Laravel acknowledge and accept a URI (Uniform Resource Identifier) along with a closure, given that it should have to be a simple and expressive way of routing. In this chapter, you will learn about the routing concept of Laravel.

All the routes in Laravel are defined within the route files that you can find in the routes sub- directory. These route files get loaded and generated automatically by Laravel framework. The application's route file gets defined in **app/Http/routes.php** file. The general routing in Laravel for each of the possible request looks something like this:

```
Route:: get ('/', function () {return 'Welcome to index';
});
Route:: post('user/dashboard', function ()
{ return 'Welcome to dashboard';
});
Route:: put('user/add', function () {
//

});
Route:: delete('post/example', function () {
//
});
```

The routing mechanism takes place in three different steps:

1. First of all, you have to create and run the root URL of your project.
2. The URL you run needs to be matched exactly with your method defined in the root.php file, and it will execute all related functions.
3. The function invokes the template files. It then calls the **view() function** with the file name located in **resources/views/**, and eliminates the file extension blade.php at the time of calling.

app/Http/routes.php

```
<?php
Route:: get ('/', function () {
return view('laravel');
});
```

resources/view/laravel.blade.php

```
<html>
<head>
  <title>Laravel5 Tutorial</title>
</head>
<body>
  <h2>Laravel5 Tutorial</h2>
  <p>Welcome to Laravel5 tutorial.</p>
```

</body>
</html>

In many cases, within your application, a situation arises when you had to capture the parameters send ahead through the URL. For using these passed parameters effectively, in Laravel, you have to change the routes.php code.

Laravel provides two ways of capturing the passed parameter:

- Required parameter
- Optional Parameter
- Required parameter

At times you had to work with a segment(s) of the URL (Uniform Resource Locator) in your project. Route parameters are encapsulated within {} (curly-braces) with alphabets inside. Let us take an example where you have to capture the ID of the customer or employee from the generated URL. Route :: get ('emp/{id}', function ($id) {   echo 'Emp '.$id;

});

- Optional Parameter

There are many parameters which do not remain present within the URL, but the developers had to use them. So such parameters get indicated by a ***?*** *(question mark sign)* following the name of the parameter. Route :: get ('emp/{desig?}', function ($desig = null) { echo $desig;

});

Route :: get ('emp/{name?}', function ($name = 'Guest') {
echo $name;
});

Step 3. Create your First Laravel Controller

In the MVC framework, the letter 'C' stands for Controller. It acts as a directing traffic between Views and Models. In this chapter, you will learn about Controllers in Laravel.
Creating a Controller

Open the command prompt or terminal based on the operating system you are using and type the following command to create controller using the Artisan CLI (Command Line Interface).

php artisan make:controller <controller-name> --plain

Replace the <controller-name> with the name of your controller. This will create a plain constructor as we are passing the argument — **plain**. If you don’t want to create a plain constructor, you can simply ignore the argument. The created constructor can be seen at **app/Http/Controllers**.

You will see that some basic coding has already been done for you and you can add your custom coding. The created controller can be called from routes.php by the following syntax.

Syntax
Route::get(‘base URI’,’controller@method’); Example

Step 1 − Execute the following command to create **UserController**. php artisan make:controller UserController --plain

Step 2 − After successful execution, you will receive the following output.

Step 3 − You can see the created controller at **app/Http/Controller/UserController.php** with some basic coding already written for you and you can add your own coding based on your need.

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

class UserController extends Controller {
   //
}
```

Controller Middleware

We have seen middleware before and it can be used with controller also. Middleware can also be assigned to controller’s route or within your controller’s constructor. You can use the middleware method to assign middleware to the controller. The registered middleware can also be restricted to certain method of the controller.

Assigning Middleware to Route

```
Route::get('profile', [
   'middleware' => 'auth',
   'uses' => 'UserController@showProfile' ]);
```

Here we are assigning auth middleware to UserController in profile route.

Assigning Middleware within Controller's constructor

```
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
use App\Http\Requests;
use App\Http\Controllers\Controller; class
UserController extends Controller {     public
function_construct() {
      $this->middleware('auth');
   }
}
```

Here we are assigning **auth** middleware using the middleware method in the **UserController** constructor.

Example

Step 1 − Add the following lines of code to the **app/Http/routes.php** file and save it.

routes.php

```
<?php
Route::get('/usercontroller/path',
[
   'middleware' => 'First',
   'uses' => 'UserController@showPath' ]);
```

Step 2 − Create a middleware called **FirstMiddleware** by executing the following line of code. php artisan make:middleware FirstMiddleware

Step 3 − Add the following code into the **handle** method of the newly created FirstMiddleware at **app/Http/Middleware.**

FirstMiddleware.php

```
<?php
namespace App\Http\Middleware;
```

```
use Closure;

class FirstMiddleware {
   public function handle($request, Closure $next) {       echo
'<br>First Middleware';       return $next($request);
   }
}
```

Step 4 − Create a middleware called **SecondMiddleware** by executing the following command.

```
php artisan make:middleware SecondMiddleware
```

Step 5 − Add the following code in the handle method of the newly created SecondMiddleware at

app/Http/Middleware.

SecondMiddleware.php

```
<?php
namespace App\Http\Middleware;
use Closure;
class SecondMiddleware {
   public function handle($request, Closure $next) {       echo
'<br>Second Middleware';       return $next($request);
   } }
```

Step 6 − Create a controller called **UserController** by executing the following line. php artisan

make:controller UserController --plain

Step 7 − After successful execution of the URL, you will receive the following output

Step 8 − Copy the following code to **app/Http/UserController.php** file.

app/Http/UserController.php

```
<?php
namespace
App\Http\Controllers;           use
Illuminate\Http\Request;       use
App\Http\Requests;
use App\Http\Controllers\Controller;
 class UserController extends Controller {     public
function__construct() {       $this-
>middleware('Second');
```

```
  }
  public function showPath(Request $request) {
    $uri = $request->path();       echo
'<br>URI: '.$uri;

    $url = $request->url();       echo '<br>';

    echo 'URL: '.$url;
    $method = $request->method();      echo
'<br>';

    echo 'Method: '.$method;
  }
}
```

Step 9 − Now launch the php's internal web server by executing the following command, if you haven't executed it yet.

```
php artisan serve
```

Step 10 − Visit the following URL.

http://localhost:8000/usercontroller/path

Step 11 − The output will appear as shown in the following image.

Restful Resource Controllers

Often while making an application we need to perform **CRUD (Create, Read, Update, Delete)** operations. Laravel makes this job easy for us. Just create a controller and Laravel will automatically provide all the methods for the CRUD operations. You can also register a single route for all the methods in routes.php file.

Example

Step 1 − Create a controller called **MyController** by executing the following command. php artisan make:controller MyController **Step 2** − Add the following code in **app/Http/Controllers/MyController.php** file. **app/Http/Controllers/MyController.php**

```
<?php
namespace
```

```
App\Http\Controllers;           use
Illuminate\Http\Request;       use
App\Http\Requests;
use App\Http\Controllers\Controller;
class MyController extends Controller { public function index() {      echo 'index';
   }
   public function create() {       echo 'create';
   }
   public function store(Request $request) {      echo 'store';
   }
   public function show($id) {        echo 'show';
   }
   public function edit($id) {        echo 'edit';
   }
   public function update(Request $request, $id) {      echo 'update';
   }
   public function destroy($id) {        echo 'destroy';
   } }
```

Step 3 − Add the following line of code in **app/Http/routes.php** file.

app/Http/routes.php

```
Route::resource('my','MyController');
```

Step 4 − We are now registering all the methods of MyController by registering a controller with resource. Below is the table of actions handled by resource controller.

Verb	Path	Action	Route Name
GET	/my	index	my.index
GET	/my/create	create	my.create
POST	/my	store	my.store
GET	/my/{my}	show	my.show
GET	/my/{my}/edit	edit	my.edit
PUT/PATCH	/my/{my}	update	my.update
DELETE	/my/{my}	destroy	my.destroy

Step 5 − Try executing the URLs shown in the following table.

URL	Description	Output Image
http://localhost:8000/my	Executes index method of MyController.php	index
http://localhost:8000/my/create	Executes create method of MyController.php	create
http://localhost:8000/my/1	Executes show method of MyController.php	show
http://localhost:8000/my/1/edit	Executes edit method of MyController.php	edit

Laravel Artisan Commands Cheat Sheet

Let’s have closer look at **Top 10 Laravel Artisan Commands** for developers while building laravel applications.

Command#	**Description & Output**
php artisan –	Check the current version of laravel installation?
php artisan	Put Laravel application in “maintenance mode”
php artisan	Display the environment laravel is running Output:
php artisan	Run Database migrations This executes all the defined migrations and create database
php artisan serve	To start Laravel project. By, default this hosts the application locally at localhost:8000 You can server with different hostname and post using “–host” and “–port” options respectively.
php artisan up	Bring UP the laravel application out of maintenance mode
php artisan auth:clear- resets	Flush the expired password tokens
php artisan cache:clear	Flush the application cache
php artisan cache:table	Create a migration for the cache database table

php artisan config:cache	Create a cache file for faster configuration loading
php artisan config:clear	Remove the configuration cache file
php artisan make:auth	Scaffold basic login and registration views and routes
php artisan make:controller TechCluesBlog	Create a new controller class using artisan command
php artisan make:migration my_blog_post	Create a new migration file. Output: Created Migration: 2019_01_27_094045_my_blog_post The new migration file 2019_01_27_094045_my_blog_post.php will be created under ../database/migrations./
php artisan make:model MyBlogPost	Create a new Eloquent model class. Output: Model created successfully. The new Model file “MyBlogPost.php” will be created in app/Models or ../app/..
php artisan route:list	List all registered routes
php artisan route:clear	Remove the route cache file
php artisan route:cache	Create a route cache file for faster route registration
php artisan vendor:publish	Publish any publishable assets from vendor packages
php artisan view:clear	Clear all compiled view files

Step 4. Create your first Laravel View with the Blade Templating Engine

In MVC framework, the letter **“V”** stands for **Views**. It separates the application logic and the presentation logic. Views are stored in **resources/views** directory. Generally, the view contains the HTML which will be served by the application.

Example

Observe the following example to understand more about Views − **Step 1**

− Copy the following code and save it at **resources/views/test.php**

```
<html>
  <body>
    <h1>Hello, World</h1>
  </body> </html>
```

Step 2 − Add the following line in **app/Http/routes.php** file to set the route for the above view.

app/Http/routes.php

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

Step 3 − Visit the following URL to see the output of the view. http://localhost:8000/test

Step 4 − The output will appear as shown in the following image.

Passing Data to Views

While building application it may be required to pass data to the views. Pass an array to view helper function. After passing an array, we can use the key to get the value of that key in the HTML file.

Example

Observe the following example to understand more about passing data to views −

Step 1 − Copy the following code and save it at **resources/views/test.php**

```
<html>
  <body>
    <h1><?php echo $name; ?></h1>
  </body>
</html>
```

Step 2 − Add the following line in **app/Http/routes.php** file to set the route for the above view.

app/Http/routes.php

```
Route::get('/test', function() {
  return view('test',['name'=>'Virat Gandhi']); });
```

Step 3 − The value of the key name will be passed to test.php file and $name will be replaced by that value.
Step 4 − Visit the following URL to see the output of the view. http://localhost:8000/test

Step 5 − The output will appear as shown in the following image.

Sharing Data with all Views

We have seen how we can pass data to views but at times, there is a need to pass data to all the views. Laravel makes this simpler. There is a method called **share()** which can be used for this purpose. The **share()** method will take two arguments, key and value. Typically **share()** method can be called from boot method of service provider. We can use any service provider, **AppServiceProvider** or our own service provider.

Example

Observe the following example to understand more about sharing data with all views −

Step 1 − Add the following line in **app/Http/routes.php** file. **app/Http/routes.php**

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

Route::get('/test2',
function() {                           return view('test2'); });
```

Step 2 − Create two view files — **test.php** and **test2.php** with the same code. These are the two files which will share data. Copy the following code in both the files. **resources/views/test.php & resources/views/test2.php**

```
<html>
  <body>
    <h1><?php echo $name; ?></h1>
  </body>
</html>
```

Step 3 − Change the code of boot method in the file **app/Providers/AppServiceProvider.php** as shown below. (Here, we have used share method and the data that we have passed will be shared with all the views.)

app/Providers/AppServiceProvider.php

```
<?php
namespace App\Providers;
use Illuminate\Support\ServiceProvider;

class AppServiceProvider extends ServiceProvider {

  /**
*  Bootstrap any application services.
    *
*  @return void
  */
  public function boot() {
    view()->share('name', 'Virat Gandhi');
  }

  /**

*  Register any application services.
    *
*  @return void
  */
  public function register() {
/
/
}
}
```

Step 4 − **Visit** the following URLs.

http://localhost:8000/test

http://localhost:8000/test2

Step 5 − The output will appear as shown in the following image.

Root directory

The Root Directory Structure of Laravel

The public directory helps in starting your Laravel project and also holds other scripts (JavaScript and CSS) as well along with images required for your project. resources. This directory is one of the most important directories inside which you will find some other subdirectories. These are: routes storage.

Directory Description

App: The app directory holds the base code for your Laravel application. bootstrap The bootstrap directory holds all the bootstrapping scripts used for your application.

Config The config directory holds all your project configuration files (.config).

Database The database directory holds your database files.

The public directory helps in starting your Laravel project and also holds other scripts public (JavaScript and CSS) as well along with images required for your project.

The resources directory holds all the Sass files, language (localization) files, templates (if resources any).

The routes directory holds all your definition files for routing such as console.php, api.php,

routes	channels.php etc.
storage	The storage directory holds your session files, cache, compiled templates as well as miscellaneous files generated by the framework.
test	The test directory holds all your test cases.
vendor	The vendor directory holds all composer dependency files.

Application directory

This is another Laravel directory which holds other subdirectories for additional purposes. These are:

Directory **Description**

Console The Console directory contains all your project artisan commands.

The Events directory hold event files that your laravel application may pop up. Events is Events used for sending messages or signals to other parts of the laravel project that any action had taken place within the project.

The Exceptions directory holds your laravel project's exception handling files which Exceptions handles all the exceptions thrown by your Laravel project.

Http: The Http directory holds different filters, requests, and controllers.

The Jobs directory holds all lineup jobs in this directory. But it does not get created Jobs initially, rather, you need to type and run this artisan command: make:job

The Listeners directory holds all your project's handler class for which are used for Listeners receiving and handling events.

The Main directory holds all the emails send by through your Laravel project, and this

Maildirectory needs to be created using the command: make:mail

The Notifications directory contains all your transactional notifications sent through your Notifications

Laravel project, and this directory needs to be created using the command: make:notification

Policies: The policies directory holds different policies for your laravel project. Providers The Providers directory is used for containing different service providers.

The Rules directory hold all the different objects related to custom validation rules, and Rules this directory needs to be created using the command:

make:rule

Namespacing

Namespaces can be defined as a class of elements in which each element has a unique name to that associated class. It may be shared with elements in other classes.

Declaration of namespace

The **use** keyword allows the developers to shorten the namespace. use <namespace-name>;

The default namespace used in Laravel is App, however a user can change the namespace to match with web application. Creating user defined namespace with artisan command is mentioned as follows − php artisan app:name SocialNet

The namespace once created can include various functionalities which can be used in controllers and various classes.

Service providers

Service providers are the central element of the initialization process where all the relevant and required code is loaded by PHP.

This includes all the essentials from the framework itself, but also any own and custom code you need to load.

Bootstrap Service Providers

During the initializing process, the code bootstraps itself. That means it gets ready to go by registering service container bindings, event listeners, middleware, configuration, and routes. Service providers are therefore a central place to set up your application.

In the **config/app.php** you will find an array of providers listing all the service provider classes that are loaded during bootstrapping.

Beware that many of the service provider classes may be so-called deferred providers. It means they are only loaded upon request, and not always included by default. Making a provider deferred works well for classes that only need to be loaded sometimes because it reduces the performance overhead and loads time of your web application.

Custom Service Providers

The code below is an example of a custom service provider. It extends the Illuminate**\Support\ServiceProvider** class which is an abstract that requires you to define at least a method called ***register.***

The purpose of the register method is to bind values in the service container.

Writing Service Providers

All service providers extend the Illuminate/Support/ServiceProvider class. Most service providers contain a register and a boot method. Within the register method. You should only blind things into the service container. You should never attempt to register any event listeners, routes, or any other piece of functionality within the register method.

The Artisan CLI can generate a new provider through the **make::provider** command. php artisan make:provider RiakServiceProvider

Copy

The Register Method

Within the register method, you should only blind thing into the service container. You should never attempt to register any event listeners, routes, or any other piece of functionality within the register method. Otherwise, you may accidentally use a service that is provided by a service provider which has not loaded yet. Let's look at this code sample

```
<?php
namespace
App\Providers; use
Riak\Connection;
use Illuminate\Support\ServiceProvider;

class RiakServiceProvider extends ServiceProvider
```

```
        {
            public function register()
            {
                $this->app->singleton(Connection::class, function ($app)
                {
                    return new Connection(config('riak'));        });
 }
 }
```
Copy

This service provider only defines a register method, and uses that method to define an implementation of Riak//Connection in the service container.

The Boot Method

So, what if we need to register a view composer within our service provider? This should be done within the boot method. This method called after all other service providers have been registered.

Let's look at a simple example.

```
<?php
namespace App\Providers;
use Illuminate\Support\ServiceProvider;
class ComposerServiceProvider extends ServiceProvider
{
    public function boot()
    {
        view()->composer('view', function () {
            //
        });
    }
}
```

Service containers

The Service Container in Laravel is a Dependency Injection Container and a Registry for the application. The advantages of using a Service Container over creating manually your objects are:

Capacity to manage class dependencies on object creation:
You define how an object should be created in one point of the application (the binding) and every time you need to create a new instance, you just ask it to the service container, and it will create it for you, along with the required dependencies For example, instead of creating objects manually with the new keyword:

//everytime we need YourClass we should pass the dependency manually
$instance = new
YourClass($dependency); Copy

Instead, you can register a binding on the Service Container:
//add a binding for the class
YourClass App::bind(
YourClass::class, function()
{

  //do some preliminary work: create the needed dependencies
$dependency = new DepClass(config('some.value'));

  //create and return the object with his
dependencies return new YourClass(
$dependency);
});

Copy and create an instance through the service container with:

//no need to create YourClass dependencies, the SC will do that for us!

$instance = App::make(

YourClass::class); Copy

With Laravel automatic dependency injection, when an interface is required in some part of the app (i.e. in a controller's constructor), a concrete class is instantiated automatically by the Service Container. Changing the concrete class on the binding, will change the concrete objects instantiated through all your app:

//every time a UserRepositoryInterface is requested, create an EloquentUserRepository

App::bind(UserRepositoryInterface::class, EloquentUserRepository::class);

Copy
//from now on, create a TestUserRepository

App::bind(UserRepositoryInterface::class,
TestUserRepository::class); Copy

Using the Service Container as a Registry

You can create and store unique object instances on the container and get them back later: using the App::instance method to make the binding, and thus using the container as a Registry.

// Create an instance.

$kevin = new User('Kevin');
// Bind it to the

```
service container.
App::instance('the-
user', $kevin);

// ...somewhere and/or in another class...

// Get back the instance

$kevin = App::make('the-user');

Copy
```

As a final note, essentially the Service Container -is- the Application object: it extends the Container class, getting all the container's functionalities.

For more about laravel, refer to https://laravel.com/docs

The link above contains all that you will need to create a full web application project.

Service Containers

The Laravel service container is a powerful tool for managing class dependencies and performing dependency injection. Dependency injection is a fancy phrase that essentially means this: class dependencies are "injected" into the class via the constructor or, in some cases, "setter" methods.

Let's look at a simple example:

```
<?php
namespace App\Http\Controllers;
use App\Http\Controllers\Controller;
use App\Repositories\UserRepository;
use App\User;

class UserController extends Controller
{
    /**
     * The user repository implementation.
     *
     * @var UserRepository
     */
    protected $users;

    /**
     * Create a new controller instance.
     *
     * @param  UserRepository  $users
     * @return void
     */
    public function __construct(UserRepository $users)
    {
        $this->users = $users;
    }

    /**
     * Show the profile for the given user.
     *
     * @param  int  $id
     * @return Response
     */
    public function show($id)
    {
        $user = $this->users->find($id);

        return view('user.profile', ['user' => $user]);
    }
}
```

In this example, the UserController needs to retrieve users from a data source. So, we will inject a service that is able to retrieve users. In this context, our UserRepository most likely uses Eloquent to retrieve user information from the database. However, since the repository is injected, we are able to easily swap it out with another implementation. We are also able to easily "mock", or create a dummy implementation of the UserRepository when testing our application.

A deep understanding of the Laravel service container is essential to building a powerful, large application, as well as for contributing to the Laravel core itself.

Contracts

Laravel's Contracts are a set of interfaces that define the core services provided by the framework. For example, a Illuminate\Contracts\Queue\Queue contract defines the methods needed for queuing jobs, while the Illuminate\Contracts\Mail\Mailer contract defines the methods needed for sending e-mail.
Each contract has a corresponding implementation provided by the framework. For example, Laravel provides a queue implementation with a variety of drivers, and a mailer implementation that is powered by SwiftMailer.
All of the Laravel contracts live in their own GitHub repository. This provides a quick reference point for all available contracts, as well as a single, decoupled package that may be utilized by package developers.
How To Use Contracts
So, how do you get an implementation of a contract? It's actually quite simple.

Many types of classes in Laravel are resolved through the service container, including controllers, event listeners, middleware, queued jobs, and even route Closures. So, to get an implementation of a contract, you can just "type-hint" the interface in the constructor of the class being resolved.

For example, take a look at this event listener:

```
<?php

namespace App\Listeners;

use App\Events\OrderWasPlaced;
use App\User;
use Illuminate\Contracts\Redis\Factory;

class CacheOrderInformation
{
   /**
    * The Redis factory implementation.
    */
   protected $redis;

   /**
    * Create a new event handler instance.
    *
    * @param  Factory  $redis
    * @return void
    */
   public function __construct(Factory $redis)
   {
      $this->redis = $redis;
   }
   /**
    * Handle the event.
    *
    * @param  OrderWasPlaced  $event
    * @return void
    */
   public function handle(OrderWasPlaced $event)
   {
      //
   }
}
```

When the event listener is resolved, the service container will read the type-hints on the constructor of the class, and inject the appropriate value. To learn more about registering things in the service container, check out its documentation.

Services

Generally speaking a Service is a class where you implement related and common functionality that you want to be used in various places. If you look into the Registrar class you mentioned, it has two methods: validator() and create(). The validator()method returns a Validator object that contains the rules for registering a new user. Note that this validation rules are specific for registering a user, you could have a different validation rule set for other operations, like letting a user to change his/her e-mail address.

The create() methods uses the User eloquent model to create a user with a minimum set of fields that are required to register. You could add more attributes to your User model, but for registering these ones will be sufficient.

If you think about it, both of these methods are very specific to a user registration process, so they should not belong to the User model, nor are candidates for a repository. This is when a service is helpful.
Another example: in a project of mine I have a Service which calculates the distance between two cities, using Google Distance Matrix API, I have something like this:

```
<?php namespace App\Services;
use GuzzleHttp\Client;
use App\City;
class DistanceService
{
  const GOOGLE_API_URL = 'http://google/api/url/here';
  public function calculate( City $from, City $to )
  {
    // 1. get cities api-comapatible name representation
    // 2. replace on the API URL
    // 3. make a request using Guzzle
    // 4. fetch and parse the response
    // 5. return the distance
  }
    /* private helper methods */
}
```

This is a very specific task, and has a somehow lengthy algorithm. This Service class encapsulates its logic, and can I use it in various places along my app.

Database

Laravel makes interacting with databases extremely simple across a variety of database backends using either raw SQL, the fluent query builder, and the Eloquent ORM. Currently, Laravel supports four databases:

- MySQL 5.6+ (Version Policy)
- PostgreSQL 9.4+ (Version Policy)
- SQLite 3.8.8+
- SQL Server 2017+ (Version Policy)

Configuration

The database configuration for your application is located at config/database.php. In this file you may define all of your database connections, as well as specify which connection should be used by default. Examples for most of the supported database systems are provided in this file.
By default, Laravel's sample environment configuration is ready to use with Laravel Homestead, which is a convenient virtual machine for doing Laravel development on your local machine. You are free to modify this configuration as needed for your local database.

SQLite Configuration

After creating a new SQLite database using a command such as touch database/database.sqlite, you can easily configure your environment variables to point to this newly created database by using the database's absolute path:

```
DB_CONNECTION=sqlite

DB_DATABASE=/absolute/path/to/database.sqlite
```

To enable foreign key constraints for SQLite connections, you should set the DB_FOREIGN_KEYS environment variable to true:

```
DB_FOREIGN_KEYS=true
```

Configuration Using URLs

Typically, database connections are configured using multiple configuration values such as host, database, username, password, etc. Each of these configuration values has its own corresponding environment variable. This means that when configuring your database connection information on a production server, you need to manage several environment variables. Some managed database providers such as Heroku provide a single database "URL" that contains all of the connection information for the database in a single string. An example database URL may look something like the following:

```
mysql://root:password@127.0.0.1/forge?charset=UTF-8
```

These URLs typically follow a standard schema convention:

```
driver://username:password@host:port/database?options
```

For convenience, Laravel supports these URLs as an alternative to configuring your database with multiple configuration options. If the url (or corresponding DATABASE_URL environment variable) configuration option is present, it will be used to extract the database connection and credential information.

Read & Write Connections

Sometimes you may wish to use one database connection for SELECT statements, and another for INSERT, UPDATE, and DELETE statements. Laravel makes this a breeze, and the proper connections will always be used whether you are using raw queries, the query builder, or the Eloquent ORM.

To see how read / write connections should be configured, let's look at this example:

```
'mysql' => [

  'read' => [
    'host' => [
      '192.168.1.1',
      '196.168.1.2',
    ],
  ],
  'write' => [
    'host' => [
      '196.168.1.3',
     ],
  ],
  'sticky'    => true,
  'driver'    => 'mysql',
  'database'  => 'database',
  'username'  => 'root',
  'password'  => '',
  'charset'   => 'utf8mb4',
  'collation' => 'utf8mb4_unicode_ci',
  'prefix'    => '',
],
```

Note that three keys have been added to the configuration array: read, write and sticky. The read and write keys have array values containing a single key: host. The rest of the database options for the read and write connections will be merged from the main mysql array.

You only need to place items in the read and write arrays if you wish to override the values from the main array. So, in this case, 192.168.1.1 will be used as the host for the "read" connection, while 192.168.1.3 will be used for the "write" connection. The database credentials, prefix, character set, and all other options in the main mysql array will be shared across both connections.

The sticky Option

The sticky option is an *optional* value that can be used to allow the immediate reading of records that have been written to the database during the current request cycle. If the sticky option is enabled and a "write" operation has been performed against the database during the current request cycle, any further "read" operations will use the "write" connection. This ensures that any data written during the request cycle can be immediately read back from the database during that same request. It is up to you to decide if this is the desired behavior for your application.

Using Multiple Database Connections

When using multiple connections, you may access each connection via the connection method on the DB facade. The name passed to the connection method should correspond to one of the connections listed in your config/database.php configuration file:

```
$users = DB::connection('foo')->select(...);
```

You may also access the raw, underlying PDO instance using the getPdo method on a connection instance:

```
$pdo = DB::connection()->getPdo();
```

Running Raw SQL Queries

Once you have configured your database connection, you may run queries using the DB facade. The DB facade provides methods for each type of query: select, update, insert, delete, and statement.

Running A Select Query

To run a basic query, you may use the select method on the DB facade:

```
<?php
namespace App\Http\Controllers;
use App\Http\Controllers\Controller;
use Illuminate\Support\Facades\DB;
class UserController extends Controller
{
    /**
     * Show a list of all of the application's users.
     *
     * @return Response
     */
    public function index()
    {
        $users = DB::select('select * from users where active = ?', [1]);

        return view('user.index', ['users' => $users]);
    }
}
```

The first argument passed to the select method is the raw SQL query, while the second argument is any parameter bindings that need to be bound to the query. Typically, these are the values of the where clause constraints. Parameter binding provides protection against SQL injection.

The select method will always return an array of results. Each result within the array will be a PHP stdClass object, allowing you to access the values of the results:

```
foreach ($users as $user) {
    echo $user->name;
}
```

Using Named Bindings

Instead of using ? to represent your parameter bindings, you may execute a query using named bindings:

```
$results = DB::select('select * from users where id = :id', ['id' => 1]);
```

Running An Insert Statement

To execute an insert statement, you may use the insert method on the DB facade. Like select, this method takes the raw SQL query as its first argument and bindings as its second argument:

```
DB::insert('insert into users (id, name) values (?, ?)', [1, 'Dayle']);
```

Running An Update Statement

The update method should be used to update existing records in the database. The number of rows affected by the statement will be returned:

```
$affected = DB::update('update users set votes = 100 where name = ?', ['John']);
```

Running A Delete Statement

The delete method should be used to delete records from the database. Like update, the number of rows affected will be returned:

```
$deleted = DB::delete('delete from users');
```

Running A General Statement

Some database statements do not return any value. For these types of operations, you may use the statement method on the DB facade:

```
DB::statement('drop table users');
```

Building a list of links

- **Build A Link Sharing Website With Laravel**

Building your own applications, even on a simple or small scale, is a great way to build your skills. This Link Sharing Website tutorial using **Laravel** will help us to do just that. We're going to cover a lot of ground as you can see in the table of contents listed below. We start right at the very basics of choosing a name for your project, and build through all the steps required to get a working application by the time you hit the last step.

I like to put things into a step by step sequence of events. This outline will provide us with each step to take, in order, when creating a basic website in Laravel.

1#Choose a project name.

2#Install Homestead.

3#Create a Laravel Application.

4#Set the Application Namespace.

5#Open the project in PHP Storm.

6#Configure base url, providers, and aliases.

7#Configure Environment Settings.

8#Create The Database.

9#Take PHP Unit for a spin.

10#Create Routes File.

11#Create Controllers With Artisan.

12#Create Models With Artisan.

13#Create Supporting Migrations.

14#Implement The Models.

15#Implement The Controllers.

16#Create Form Requests.

17#Create Associated Forms.

18#Create A Master Layout.

19#Create Remaining Blade Views.

20#Include a Helper Class.

21#Implement Post Votes With Ajax.

22#Set redirectTo for Laravel Auth System.

23#Screenshots of our App.

#Choose a project name

We will go soup to nuts here, with everything built on top of Laravel Homestead. It's the best way to have a reliable and flexible development platform right on your own machine. If you'd like to follow along, do **install Laravel Homestead** first. We'll call our project angleslash, and we'll want to be able to simpy visit http://angleslash.dev in our browser, and have everything work.

#Install Homestead

Getting Homestead running is a tutorial in itself. Just **read the documentation** to get up and running, or visit our article here at Vegibit that talks about configuring Homestead.

We'll need to configure our **Homestead.yaml** file to add our new site to the homestead server. This is pretty easy, just make sure not to use any tab characters in the file, otherwise vagrant will bug out on you. Our current Homestead.yaml file has these entries in the sites section.

```
sites:
    - map: homestead.app
      to: /home/vagrant/Code/laravel/public
    - map: larabook.dev
      to: /home/vagrant/Code/larabook/public
    - map: angleslash.dev
      to: /home/vagrant/Code/angleslash/public
```

note: If you already have homestead running, you can still simply edit the Homestead.yaml, then run: vagrant provision, and you'll be good to go. In addition, you'll need to set your hosts file to add an entry for your new project. For windows this is located at C:\Windows\System32\drivers\etc\hosts Our current hosts file looks like this:

```
#homestead
```

```
2 192.168.10.10   homestead.app
3 192.168.10.10   larabook.dev
4 192.168.10.10   angleslash.dev
```

You can see that this homestead server has 3 different projects running all at one time. We have a base homestead.app project for various testing, we installed the **larabook application** from Jeffrey Way's Laracast series, and now we have our new social link sharing website, angleslash.dev. Homestead really does provide a nice and convenient way to work on multiple projects at once. If you're developing multiple websites, it is a must.

#Create the angleslash app

Now that we have everything ready to go, we can start building the project. The first thing we'll need to do is ssh into our server.

```
vagrant ssh
```

Once we are logged in to our server, we can easily create a new project with Composer.

```
vagrant@homestead:~/Code composer create-project laravel/laravel angleslash --prefer-dist
```

#Set the Application Namespace

We can easily set the namespace for our application with Artisan, so let's do so.

```
vagrant@homestead:~/Code/angleslash$ php artisan app:name angleslash
```

You should see the result, **Application namespace set!**

#Open the project in PHP Storm

Yes it's a paid solution, but it's the best. It also helps to really see how all of the code of a framework ties together. So if you don't have it yet, go buy yourself a copy! We'll add barryvdh's ide helper since it is so fantastic to help us with code completion.

```
vagrant@homestead:~/Code/angleslash$ composer require barryvdh/laravel-ide-helper
```

Let's add a few more packages that may be helpful while we are at it.

```
vagrant@homestead:~/Code/angleslash$ composer require guzzlehttp/guzzle
vagrant@homestead:~/Code/angleslash$ composer require illuminate/html
vagrant@homestead:~/Code/angleslash$ composer require laravel/cashier
```

Notice that since we simply used Composer to add the requirements, we don't even have to manually edit or futz around with the composer.json file. All of that gets taken care of for us by composer which is nice.

#Configure The app.php file

Once the dependencies are set up the first place we'll usually head to is the **app.php** file found in the config directory of the root namespace. In here we can configure some of the basic settings that any application will need.

First, we'll configure our base url.

```
1 'url' => 'http://angleslash.dev',
```

Next, we will populate the providers array with some new entries.

```
1 'Barryvdh\LaravelIdeHelper\IdeHelperServiceProvider',
2 'Illuminate\Html\HtmlServiceProvider',
3 'Laravel\Cashier\CashierServiceProvider',
```

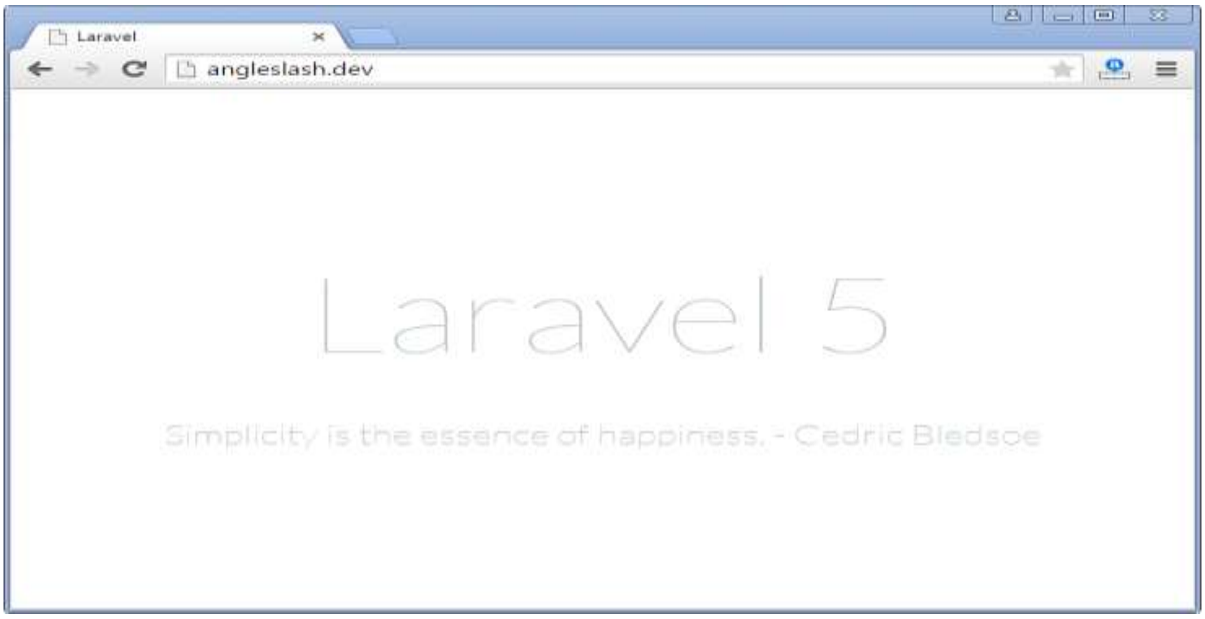
Finally we will configure the aliases array.

```
1 'Html'     => 'Illuminate\Html\HtmlFacade',
2 'Form'     => 'Illuminate\Html\FormFacade',
```

This will allow us to continue to use the original HTML and Form helpers from earlier versions of Laravel. For good measure, lets now do a **composer update**, then generate the docs for the ide helper. By completing the ide-helper:generate command, our code editor will now have intellisense like auto complete for Laravel. It's a fantastic feature.

```
vagrant@homestead:~/Code/angleslash$ composer update
vagrant@homestead:~/Code/angleslash$ php artisan ide-helper:generate
```

We've done a fair amount of configuration, and we have a pretty nice base install of the Laravel framework going. Just to confirm, we'll visit http://angleslash.dev/, and yes, we see the friendly splash page indicating that we have a successful installation of Laravel to work with.



In fact, the tutorial up until this point is a great workflow checklist no matter what type of project you might be creating. Jeez, I might need to bookmark this post!

#Configure Environment Settings

In the root namespace, we can open the .env file and set some information. Really the only thing we are worried about right now in this development environment is to set the database name. We'll use angleslash as the database name for this project. All other options in the .env file will be left at defaults for now.

```
DB_HOST=localhost
DB_DATABASE=angleslash
DB_USERNAME=homestead
DB_PASSWORD=secret
```

#Create The Database

Our application is going to need a database to work with. Since we have multiple projects going on our homestead server, we'll need to create a new database to work with. Recall that the user for mysql in homestead is *homestead* and the password is *secret*. Lets create this database from the mysql terminal now.

```
vagrant@homestead:~$ mysql -u homestead -p
Enter password: secret

mysql> create database angleslash;
Query OK, 1 row affected (0.00 sec)
```

```
mysql> show databases;

+--------------------+
| Database           |
+--------------------+
| information_schema |
| angleslash         |
| homestead          |
| mysql              |
| performance_schema |
+--------------------+
5 rows in set (0.00 sec)
mysql>exit
```

Perfect! With that, we now have a database ready to use, and we can move on.

#Run Your First PHP Test

From the terminal, we can run the example test that comes with Laravel. This is the command to run it.

```
vagrant@homestead:~/Code/angleslash$ vendor/bin/phpunit
```

```
PHPUnit 4.6.7 by Sebastian Bergmann and contributors.
Configuration read from /home/vagrant/Code/angleslash/phpunit.xml
Time: 4.61 seconds, Memory: 13.50Mb
OK (1 test, 1 assertion)
```

The test above simply ran the ExampleTest.php in the tests directory. Let's take a look at the code involved.

```php
<?php

class ExampleTest extends TestCase {

	/**
	 * A basic functional test example.
	 *
	 * @return void
	 */
	public function testBasicExample()
	{
		$response = $this->call('GET', '/');

		$this->assertEquals(200, $response->getStatusCode());
	}

}
```

It just makes sure that visiting the homepage gives back a 200 ok response.

#Create The Routes File

The routes file in Laravel gives us a high level overview of how our application will function. We can see all of the endpoints that need to respond to requests, as well as which controllers will be in use. Here is the routes file we have for our link sharing website. After we quickly review the routes file, we’ll then build out the link sharing website according to how we have our routes file configured.

Routes File

```
<?php

// We use the Laravel built in Auth
Route::controllers([
   'auth' => 'Auth\AuthController',
   'password' => 'Auth\PasswordController',
]);
//---------------------------------------//

// Display users, subs and front page
Route::get('/', 'PostController@frontpage');
Route::get('u/{user}', 'UserController@show');
Route::get('r/{sub}', 'SubController@show');
//---------------------------------------//

// Checking if the user is signed in (using AJAX)
Route::get('authcheck', function () {
   return json_encode(Auth::check());
});
//---------------------------------------//

// Creating and storing a new sub
Route::get('sub/new', [
   'middleware' => 'auth',
   'uses' => 'SubController@displayform'
]);

Route::post('sub/new', [
   'middleware' => 'auth',
   'uses' => 'SubController@storesub'
]);
//---------------------------------------//

// Creating and storing a new post / link
Route::get('post/new', [
   'middleware' => 'auth',
   'uses' => 'PostController@displayform',
]);

Route::post('post/new', [
   'middleware' => 'auth',
    'uses' => 'PostController@storepost'
]);
//----------------------------------------//

// Voting on posts / links
Route::post('vote', [
    'middleware' => 'auth',
    'uses' => 'VoteController@vote'
]);
```

#Create Controllers With Artisan

We'll next use **Artisan** to generate the controllers we'll need in our application. I'm trying to become as familiar as possible with Artisan, since automatic code generation sounds like a great deal to me. First off, let's just take a quick look at all of the code generation options available to us. We can do this by piping the output of php artisan to the grep command while searching for *make*.

```
vagrant@homestead:~/Code/angleslash$ php artisan | grep make
  make
   make:command      Create a new command class
   make:console      Create a new Artisan command
   make:controller   Create a new resource controller class
   make:event        Create a new event class
   make:middleware   Create a new middleware class
   make:migration    Create a new migration file
   make:model        Create a new Eloquent model class
   make:provider     Create a new service provider class
   make:request      Create a new form request class
```

This is great! Artisan provides the ability to create new command classes, resources, events, middleware, migrations, service providers, and also the Form Requests which people have been raving about. Let's quickly create all of the controllers we'll need for our application.

```
vagrant@homestead:~/Code/angleslash$ php artisan make:controller UserController
Controller created successfully.
vagrant@homestead:~/Code/angleslash$ php artisan make:controller PostController
Controller created successfully.
vagrant@homestead:~/Code/angleslash$ php artisan make:controller SubController
Controller created successfully.
vagrant@homestead:~/Code/angleslash$ php artisan make:controller VoteController
Controller created successfully.
```

All of our controllers are now created. They are currently empty, but we are making progress with creating the skeleton of the application. Let us now create the Models.

#Create Models With Artisan

We are going to use the built in User Model that ships with Laravel, so we don't need to create that. We will modify it slightly, and add some Eloquent Relationships to it a little later. We do need some models to handle our Posts, Post Votes, and Subs. Let's create those now again using Artisan. Subs are just categories, like a subreddit.

```
vagrant@homestead:~/Code/angleslash$ php artisan make:model Post
Model created successfully.
Created Migration: 2015_06_01_175305_create_posts_table
vagrant@homestead:~/Code/angleslash$ php artisan make:model PostVote
Model created successfully.
Created Migration: 2015_06_01_175322_create_post_votes_table
vagrant@homestead:~/Code/angleslash$ php artisan make:model Sub
Model created successfully.
Created Migration: 2015_06_01_175337_create_subs_table
vagrant@homestead:~/Code/angleslash$ php artisan make:migration foreign_keys
Created Migration: 2015_06_01_181451_foreign_keys
```

Note: When we create each Model, Laravel automatically creates a migration for us. Since we have our migration files set up, let's populate our migrations with the Schema we will need to support our basic link sharing website.

#Create Supporting Migrations

create_posts_table.php

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreatePostsTable extends Migration
{
  public function up()
  {
    Schema::create('posts', function (Blueprint $table) {
      $table->increments('id');
      $table->timestamps();
      $table->string('title', 100);
      $table->string('url', 2083);
      $table->integer('sub_id')->unsigned();
      $table->integer('user_id')->unsigned();
    });
  }

  public function down()
```

```
    {
        Schema::drop('posts');
    }
}
```

create_postvotes_table.php

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreatePostVotesTable extends Migration
{
    public function up()
    {
        Schema::create('post_votes', function (Blueprint $table) {
            $table->increments('id');
            $table->timestamps();
            $table->string('type', 4);
            $table->integer('user_id')->unsigned();
            $table->integer('post_id')->unsigned();
        });
    }

    public function down()
    {
        Schema::drop('post_votes');
    }
}
```

create_subs_table.php

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class CreateSubsTable extends Migration
{
    public function up()
    {
        Schema::create('subs', function (Blueprint $table) {
            $table->increments('id');
            $table->timestamps();
            $table->string('name', 20);
            $table->integer('owner_id')->unsigned();
        });
    }

    public function down()
    {
        Schema::drop('subs');
    }
}
```

create_foreign_keys.php

```
<?php

use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;

class ForeignKeys extends Migration
{
    public function up()
    {
        Schema::table('posts', function (Blueprint $table) {
            $table->foreign('sub_id')->references('id')->on('subs')
                ->onDelete('restrict')
                ->onUpdate('restrict');
        });
        Schema::table('posts', function (Blueprint $table) {
            $table->foreign('user_id')->references('id')->on('users')
                ->onDelete('restrict')
                ->onUpdate('restrict');
        });
        Schema::table('subs', function (Blueprint $table) {
            $table->foreign('owner_id')->references('id')->on('users')
                ->onDelete('restrict')
                ->onUpdate('restrict');
        });
        Schema::table('post_votes', function (Blueprint $table) {
            $table->foreign('user_id')->references('id')->on('users')
                ->onDelete('restrict')
                ->onUpdate('restrict');
        });
        Schema::table('post_votes', function (Blueprint $table) {
            $table->foreign('post_id')->references('id')->on('posts')
                ->onDelete('restrict')
                ->onUpdate('restrict');
        });
    }

    public function down()
    {
        Schema::table('posts', function (Blueprint $table) {
            $table->dropForeign('posts_sub_id_foreign');
        });
        Schema::table('posts', function (Blueprint $table) {
            $table->dropForeign('posts_user_id_foreign');
        });
        Schema::table('subs', function (Blueprint $table) {
            $table->dropForeign('subs_owner_id_foreign');
        });
        Schema::table('post_votes', function (Blueprint $table) {
            $table->dropForeign('post_votes_user_id_foreign');
        });
        Schema::table('post_votes', function (Blueprint $table) {
            $table->dropForeign('post_votes_post_id_foreign');
        });
    }
}
```

With our migration files now created, let's run the migrations with Artisan to create the tables in the database we'll need.

```
vagrant@homestead:~/Code/angleslash$ php artisan migrate
Migration table created successfully.
Migrated: 2014_10_12_000000_create_users_table
Migrated: 2014_10_12_100000_create_password_resets_table
Migrated: 2015_06_01_175305_create_posts_table
Migrated: 2015_06_01_175322_create_post_votes_table
Migrated: 2015_06_01_175337_create_subs_table
Migrated: 2015_06_01_181451_foreign_keys
```

If we log in to mysql and issue the show tables command on the database in question, we'll see that our migrations did a great job of creating all the tables we'll need.

```
mysql> show tables;
+----------------------+
| Tables_in_angleslash |
+----------------------+
| migrations           |
| password_resets      |
| post_votes           |
| posts                |
| subs                 |
| users                |
+----------------------+
6 rows in set (0.00 sec)
```

#Implement Our Models

We were able to use Artisan to create the boilerplate for our models. We still need to fill in the logic on our own. Just like filling in the code for Schema, this requires some thought by the developer. We will make use **Laravel hasMany** and **Laravel belongsTo** relationships. This is how we might do that.

User Model

```
<?php namespace angleslash;

use Illuminate\Auth\Authenticatable;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Auth\Passwords\CanResetPassword;
use Illuminate\Contracts\Auth\Authenticatable as AuthenticatableContract;
use Illuminate\Contracts\Auth\CanResetPassword as CanResetPasswordContract;

class User extends Model implements AuthenticatableContract, CanResetPasswordContract
{

   use Authenticatable, CanResetPassword;

   protected $table = 'users';
   protected $fillable = ['name', 'email', 'password'];
   protected $hidden = ['password', 'remember_token'];

   public function posts()
   {
      return $this->hasMany('angleslash\Post');
   }

   public function subs()
   {
      return $this->hasMany('angleslash\Sub');
   }

   public function postvotes()
   {
      return $this->hasMany('angleslash\PostVote');
   }
}
```

Sub Model

```
<?php namespace angleslash;

use Illuminate\Database\Eloquent\Model;

class Sub extends Model
{
   protected $table = 'subs';
```

```
    protected $fillable = ['name', 'owner_id'];

    public function posts()
    {
        return $this->hasMany('angleslash\Post');
    }

    public function owner()
    {
        return $this->belongsTo('angleslash\User');
    }
}
```

Post Model

```
<?php namespace angleslash;

use Illuminate\Database\Eloquent\Model;

class Post extends Model
{
    protected $table = 'posts';
    protected $fillable = ['title', 'url', 'sub_id', 'user_id'];

    public function user()
    {
        return $this->belongsTo('angleslash\User');
    }

    public function sub()
    {
        return $this->belongsTo('angleslash\Sub');
    }

    public function votes()
    {
        return $this->hasMany('angleslash\PostVote');
    }
}
```

Post Vote Model

```
<?php namespace angleslash;

use Illuminate\Database\Eloquent\Model;

class PostVote extends Model
{
    protected $table = 'post_votes';
    protected $fillable = ['type', 'user_id', 'post_id'];

    public function post()
    {
        return $this->belongsTo('angleslash\Post');
    }

    public function user()
    {
        return $this->belongsTo('angleslash\User');
    }
}
```

#Implement Our Controllers

Just like our Models, we need to still fill in the logic of our controllers. We have to account for the User, Sub, Post, and Vote Controllers. Let's do that here.

User Contoller

```
<?php namespace angleslash\Http\Controllers;

use angleslash\Http\Requests;
use angleslash\Http\Controllers\Controller;

use Illuminate\Http\Request;
use angleslash\User;

class UserController extends Controller
{
    public function show($name)
    {
        $user = User::where('name', $name)->firstOrFail();

        return view('profile')
            ->with('title', $user->name)
            ->with('user', $user);
    }
}
```

Sub Contoller

```
<?php namespace angleslash\Http\Controllers;

use angleslash\Http\Requests;
use angleslash\Http\Controllers\Controller;
use angleslash\Http\Requests\SubFormRequest;
use angleslash\Sub;

use Illuminate\Http\Request;

class SubController extends Controller
```

```
{
    public function show($name)
    {
        $sub = Sub::where('name', $name)->firstOrFail();

        return view('sub')
            ->with('sub', $sub->name)
            ->with('posts', $sub->posts()->paginate(15));
    }

    public function displayform()
    {
        return view('forms.createsub')
            ->with('title', 'Create Sub');
    }

    public function storesub(SubFormRequest $request)
    {
        $sub = new Sub;
        $sub->name = $request->get('name');
        $sub->owner_id = \Auth::id();
        $sub->save();

        return \Redirect::to('r/' . $request->get('name'));
    }
}
```

Post Controller

```
<?php namespace angleslash\Http\Controllers;

use angleslash\Http\Requests;
use angleslash\Http\Controllers\Controller;
use angleslash\Sub;
use angleslash\Post;
use angleslash\PostVote;
use angleslash\Http\Requests\PostFormRequest;

use Illuminate\Http\Request;

class PostController extends Controller
{
  public function show($sub, $postId)
  {
    $sub = Sub::where('name', $sub)->firstOrFail();
    $post = Post::find($postId);

    return view('post')
      ->with('title', $post->title)
      ->with('sub', $post->sub->name)
      ->with('post', $post);
  }

  public function frontpage()
  {
    return view('sub')
      ->with('title', 'Front Page')
      ->with('posts', Post::paginate(15));
  }

  public function displayform()
  {
    return view('forms.submit')
      ->with('title', 'New Post');
  }

  public function storepost(PostFormRequest $request)
  {
    $post = new Post;

    $post->title = $request->get('title');
    $post->url = $request->get('url');
    $post->sub_id = Sub::where('name', $request->get('sub'))->first()->id;
    $post->user_id = \Auth::id();
    $post->save();
    return \Redirect::to('/');
  }}
```

Vote Controller

```php
<?php namespace angleslash\Http\Controllers;

use angleslash\Http\Requests;
use angleslash\Http\Controllers\Controller;
use angleslash\PostVote;

use Illuminate\Http\Request;

class VoteController extends Controller
{
    public function vote()
    {
        $class = \Input::get('class');
        $postId = \Input::get('postId');
        $previousVote = PostVote::where('user_id', \Auth::id())->where('post_id', $postId)->first();
        $isUpvote = str_contains($class, 'up');

        // If there is a vote by the same user on the same post
        if (!is_null($previousVote)) {
            if ($isUpvote) {
                if ($previousVote->type === 'up') {
                    // Cancel out previous upvote
                    $previousVote->delete();
                } else {
                    $previousVote->update(['type' => 'up']);
                }
            } else {
                if ($previousVote->type === 'down') {
                    // Cancel out previous downvote
                    $previousVote->delete();
                } else {
                    $previousVote->update(['type' => 'down']);
                }
            }
        } else {

            // Create a new vote
            PostVote::create([
                'type' => $isUpvote ? 'up' : 'down',
                'user_id' => \Auth::id(),
                'post_id' => $postId
            ]);
        }
    }
}
```

#Create Form Requests

We're making some serious progress here. In the above section when creating our controllers, you may have noticed that we used type hinting in the store methods for both the Sub and Post controllers. This is a really slick way to complete validation with the awesome new feature in Laravel, Form Requests. The way it works is, we can simply create a new Request using Artisan, and then fill in the **validation rules** for the request. Once this is done, we can simply pass in the request to the associated method where we need to validate data. Let's create our Form Request for both the Sub and Post now.

```
vagrant@homestead:~/Code/angleslash$ php artisan make:request SubFormRequest
Request created successfully.
vagrant@homestead:~/Code/angleslash$ php artisan make:request PostFormRequest
Request created successfully.
```

SubFormRequest

```
<?php namespace angleslash\Http\Requests;

use angleslash\Http\Requests\Request;

class SubFormRequest extends Request
{

    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'name' => 'required|min:3|max:20|alpha_dash|unique:subs'
        ];
    }
}
```

PostFormRequest

```
<?php namespace angleslash\Http\Requests;

use angleslash\Http\Requests\Request;

class PostFormRequest extends Request
{

    /**
     * Determine if the user is authorized to make this request.
     *
     * @return bool
     */
    public function authorize()
    {
        return true;
    }

    /**
     * Get the validation rules that apply to the request.
     *
     * @return array
     */
    public function rules()
    {
        return [
            'title' => 'required|max:100',
            'url' => 'required|max:2083|active_url',
            'sub' => 'required|exists:subs,name'
        ];
    }
}
```

All we had to do was provide the rules for our validation. Form Requests are a really fantastic new feature of Laravel.

#Create Associated Forms

We need to create two new forms for our application. One for creating a new Sub, or category, and another for creating a new Post or Link. Since we are using Laravel's built in forms for Registering and Logging in, we don't even need to worry about those. If you want to dig into creating forms, you can also check out this **laravel forms tutorial**. Here is the code for both of our forms.

createsub.blade.php

```
@extends('default')

@section('content')

    {!! Form::open(['url' => 'sub/new']) !!}

```

```
  <div class="col-sm-8 col-sm-offset-2">
    <p class="text-center alert alert-info"><span class="glyphicon glyphicon-ok"></span>
<b>Create</b> a new
      sub</p>
  </div>

  <div class="col-sm-8 col-sm-offset-2">
    @if($errors->first('name'))
      {!! $errors->first('name', '<div class="alert alert-warning"><b>:message</b></div>') !!}
      <?php $name = 'has-error'; ?>
    @endif

    <div class="form-group {!! $name or '' !!}">
      {!! Form::text('name', $value = null, ['class' => 'form-control input-lg', 'placeholder' => 'Enter
the name of the sub, then click Go']) !!}
    </div>

    {!! Form::submit('Go!', ['class' => 'btn btn-lg btn-block btn-info']) !!}
  </div>

  {!! Form::close() !!}

@endsection
```

submit.blade.php

```
@extends('default')

@section('content')

  {!! Form::open(['url' => 'post/new']) !!}

  <div class="col-sm-8 col-sm-offset-2">
    <p class="text-center alert alert-info"><span class="glyphicon glyphicon-ok"></span>
<b>Submit</b> a new
      link</p>
  </div>

  <div class="col-sm-8 col-sm-offset-2">
    @foreach ($errors->all(':message') as $message)
      <div class="alert alert-warning"><b>{!! $message !!}</b></div>
      @if(str_contains($message,'title'))
        <?php $title = 'has-error'; ?>
      @endif
      @if(str_contains($message,'url'))
        <?php $url = 'has-error'; ?>
      @endif
      @if(str_contains($message,'sub'))
        <?php $sub = 'has-error'; ?>
      @endif
```

```
        @endforeach
        <div class="form-group {!! $title or '' !!}">
            {!! Form::text('title', $value = null, ['class' => 'form-control input-lg', 'placeholder' => 'Enter the title here']) !!}
        </div>

        <div class="form-group {!! $url or '' !!}">
            {!! Form::text('url', $value = null, ['class' => 'form-control input-lg', 'placeholder' => 'http://example.com']) !!}
        </div>

        <div class="form-group {!! $sub or '' !!}">
            {!! Form::text('sub', $value = null, ['class' => 'form-control input-lg', 'placeholder' => 'Name of sub here']) !!}
        </div>

        {!! Form::submit('Go!', ['class' => 'btn btn-lg btn-block btn-info']) !!}
    </div>

    {!! Form::close() !!}
@endsection
```

Note: We did add a small bit of logic to include the ability to add an error class to whatever form field fails to validate. For example, if you enter an invalid URL and submit the form, when it comes back to the form with errors, not only will the application spell out the problem for you, but the URL form field will have a red outline around it via the Bootstrap has-error class.

#Create A Master Layout

Blade is growing on me in a major way. The more you use it, the more you love it. In this master layout for our link sharing website application, we'll include all of the asset resources we need to support us. Here is our master layout file.

default.blade.php

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <script src="http://code.jquery.com/jquery-2.1.4.min.js"></script>

    <!-- Include Bootstrap -->
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/css/bootstrap.min.css">
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/css/bootstrap-theme.min.css">
    <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.4/js/bootstrap.min.js"></script>
    <script src="/js/main.js"></script>
    <link rel="stylesheet" href="/css/app.css">
    <meta name="csrf-token" content="{{ csrf_token() }}">

    <title>{{ $sub or 'Welcome' }} | Angleslash</title>
</head>

<body class="container">
<nav class="navbar navbar-default">
    <div class="container-fluid">
        <div class="navbar-header">
            <button type="button" class="navbar-toggle collapsed" data-toggle="collapse"
                    data-target="#bs-example-navbar-collapse-1">
                <span class="sr-only">Toggle Navigation</span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
                <span class="icon-bar"></span>
            </button>
            <a class="navbar-brand" href="{{ url('/') }}">&lt;Angleslash&gt;</a>
        </div>

        <div class="collapse navbar-collapse" id="bs-example-navbar-collapse-1">
            <ul class="nav navbar-nav">

            </ul>

            <ul class="nav navbar-nav navbar-right">
                @if (Auth::guest())
                    <li><a href="{{ url('/auth/login') }}">Login</a></li>
                    <li><a href="{{ url('/auth/register') }}">Register</a></li>
                @else
                    <li><a href="/post/new">Submit Link</a></li>
                    <li><a href="/sub/new">Create Sub</a></li>
                    <li class="dropdown">
                        <a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button"
                           aria-expanded="false">{{ Auth::user()->name }} <span class="caret"></span></a>
                        <ul class="dropdown-menu" role="menu">
                            <li><a href="{{ url('/auth/logout') }}">Logout</a></li>
                        </ul>
                    </li>
                @endif
            </ul>
        </div>
    </div>
</nav>
@yield('content')

<div id="modal" class="modal fade">

    <div class="modal-dialog">
```

```
        <div class="modal-content">
            <div class="modal-header">
                <button type="button" class="close" data-dismiss="modal" aria-label="Close"><span
                        aria-hidden="true">&times;</span></button>
                <h4 class="modal-title">Boo Yeah!</h4>
            </div>
            <div class="modal-body">
                <h2>Join Angleslash</h2>

                <p class="lead">In order to vote, you must be signed in!</p>

                <div class="lead text-center"><a href="{{ url('/signup') }}">Register</a> or <a
                        href="{{ url('/signin') }}">Login</a></div>
            </div>
            <div class="modal-footer">
                &lt;Angleslash&gt;
            </div>
        </div>
        <!-- /.modal-content -->
    </div>
    <!-- /.modal-dialog -->
</div>
<!-- /.modal -->

</body>
</html>
```

#Create Sub, Post, and User Profile Blade Views

sub.blade.php

```
@extends('default')

@section('content')
    <div>
        @if ($posts->count() > 0)
            @foreach ($posts as $post)
                @include('snippets.post', ['post' => $post])
            @endforeach
        @else
            <p>Nobody has submmitted a post yet, looks like you will be the first!</p>
        @endif
    </div>
@endsection
```

post.blade.php

```
@extends('default')

@section('content')
```

```
<div class="container">

   @include('snippets.post', $post)

</div>
@endsection
```

profile.blade.php

```
@extends('default')

@section('content')

  <!--The user's posts-->
  <div class="col-lg-8">
     @foreach($user->posts()->paginate(15) as $post)
        @include('snippets.post', array('post' => $post))
     @endforeach

  </div>

  <!--The user profile-->
  <div class="col-lg-4">
     <div class="alert bg-success">
        <h3>User Profile</h3>

        <p>{{ $user->name }} has been a user for {{ Helper::timeAgo($user->created_at, 'user') }} and
has
           submitted {{ $user->posts->count() }} posts so far</p>
     </div>
  </div>

@endsection
```

snippets/post.blade.php

```
<?php
$vote = angleslash\PostVote::where('user_id', Auth::id())->where('post_id', $post->id)->first();
$type = null;

if (!is_null($vote)) {
  $type = $vote->type;
}
?>
<div class="panel panel-default {{ $post->id }}">
  <div class="panel-body bg-info">
     <div class="col-xs-1">
        <span class="lead glyphicon glyphicon-menu-up vote {{ $type === 'up' ? 'active' : '' }}"
             aria-hidden="true"></span>
     </div>
```

```
    <div class="col-xs-11">
      <a href="{{ $post->url }}">
        <h3>{{{ $post->title }}}</h3>
      </a>
    </div>
    <div class="col-xs-1">
      <span class="lead glyphicon glyphicon-menu-down vote {{ $type === 'down' ? 'active' : '' }}"
            aria-hidden="true"></span>
    </div>
    <div class="col-xs-11">
      <div class="votes">{{ $post->votes()->count() }} votes so far</div>
    </div>
    <div class="col-xs-12">
      <span class="pull-right">
      submitted {{ Helper::timeAgo($post->created_at) }} ago by
      <a href="/u/{{ $post->user->name }}">
        {{ $post->user->name }}
      </a>
      to
      <a href="/r/{{ $post->sub->name }}">
        {{ $post->sub->name }}
      </a>
      </span>
    </div>
  </div>

</div>
```

#Implement a Helper Class

You may have noticed in our view file we referenced a method like so Helper::timeAgo(). In order to make that available to us in our views, we'll need to include this class in our application. This is how we did it here. In the app folder, where the models live, simply add a file named Helper.php. Place this code inside of it.

Helper.php

```
<?php

class Helper {

  public static function timeAgo($datetime, $type = 'post')
  {
    $now   = new DateTime;
    $ago   = new DateTime($datetime);
    $diff  = $now->diff($ago);
    $result = '';

```

```php
        $diff->w = floor($diff->d / 7);
        $diff->d -= $diff->w * 7;

        $string = [
            'y' => 'year',
            'm' => 'month',
            'w' => 'week',
            'd' => 'day',
            'h' => 'hour',
            'i' => 'minute',
            's' => 'second',
        ];
        foreach ($string as $k => &$v)
        {
            if ($diff->$k)
            {
                $v = $diff->$k . ' ' . $v . ($diff->$k > 1 ? 's' : '');
            }
            else
            {
                unset($string[$k]);
            }
        }

        $string = array_slice($string, 0, 1);
        if ($type === 'post')
        {
            $result = $string ? implode(', ', $string) . ' ago' : 'just now';
        }
        else if ($type === 'user')
        {
            $result = $string ? implode(', ', $string) . '' : 'a nanosecond';
        }

        return $result;
    }
}
```

Modify the autoload block in composer.json to include this file.

```json
            "autoload": {
                        "classmap": [
                                    "database"
                        ],
                        "psr-4": {
                                    "angleslash\\": "app/"
                        },
        "files": [
            "app/Helper.php"
        ]
            },
```

Finally, run composer dump.

```
vagrant@homestead:~/Code/angleslash$ composer dump
Generating autoload files
```

The helper is now available to us in our views.

#Implement Post Votes With Ajax

We need a bit of JavaScript to provide for the ability to **vote on Posts** via ajax. Here is the code we use to accomplish this.

main.js

```
jQuery(document).ready(function($){

  // Check if a user is signed in
  var isSignedIn = $.ajax({
    url: '/authcheck',
    method: 'get',
    async: false
  }).responseText === 'true' ? true : false;

  $.ajaxSetup({
    headers: {
      'X-CSRF-Token': jQuery('meta[name="csrf-token"]').attr('content')
    }
  });

  // handle votes
  jQuery('.vote').click(function () {
    if (isSignedIn) {
      var postId = $(this).closest('.panel').attr('class').split(' ')[2];
      $(this).toggleClass('active');

      if ($(this).hasClass('glyphicon-menu-up')) {
        jQuery('.post-' + postId + ' .vote.glyphicon-menu-down').removeClass('active');
      } else if ($(this).hasClass('glyphicon-menu-down')) {
        jQuery('.post-' + postId + ' .vote.glyphicon-menu-up').removeClass('active');
      }

      $.ajax({
        url: '/vote',
        method: 'post',
        data: {
          'class': $(this).attr('class'),
          'postId': postId
        }
      });
    } else {
      jQuery('#modal').modal();
```

```
        }
    });
});
```

#Set redirectTo for Laravel Auth System

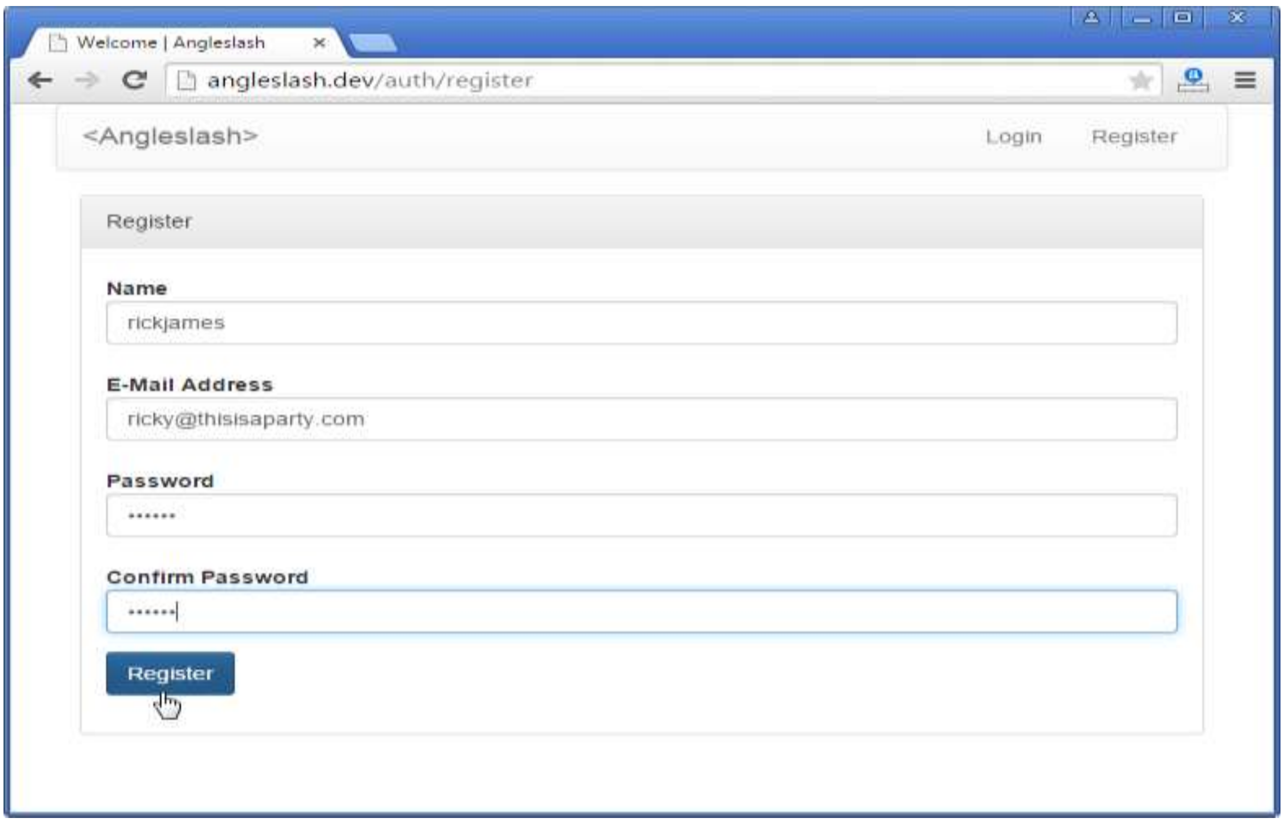
Laravel's built in registration, login, and password reset features are really fantastic. By default, these features redirect you to /home which might be the dashboard of your application. In our case, we simply use / as the "home" of our application. To set this, simply open both the AuthController.php and PasswordController.php and add this one line.

```
protected $redirectTo = '/';
```

#Screenshots of Angleslash

We've done a lot of work up until this point. We are now ready to take our link sharing website application for a spin. We'll make use of a handful of excellent links to test our link sharing website application. Let's try **http://laravel.com**, **https://leanpub.com/easyecommerce**, **https://laravel-news.com/**, and **http://www.easylaravelbook.com/** Let's check it out!

Registering A New User



Register

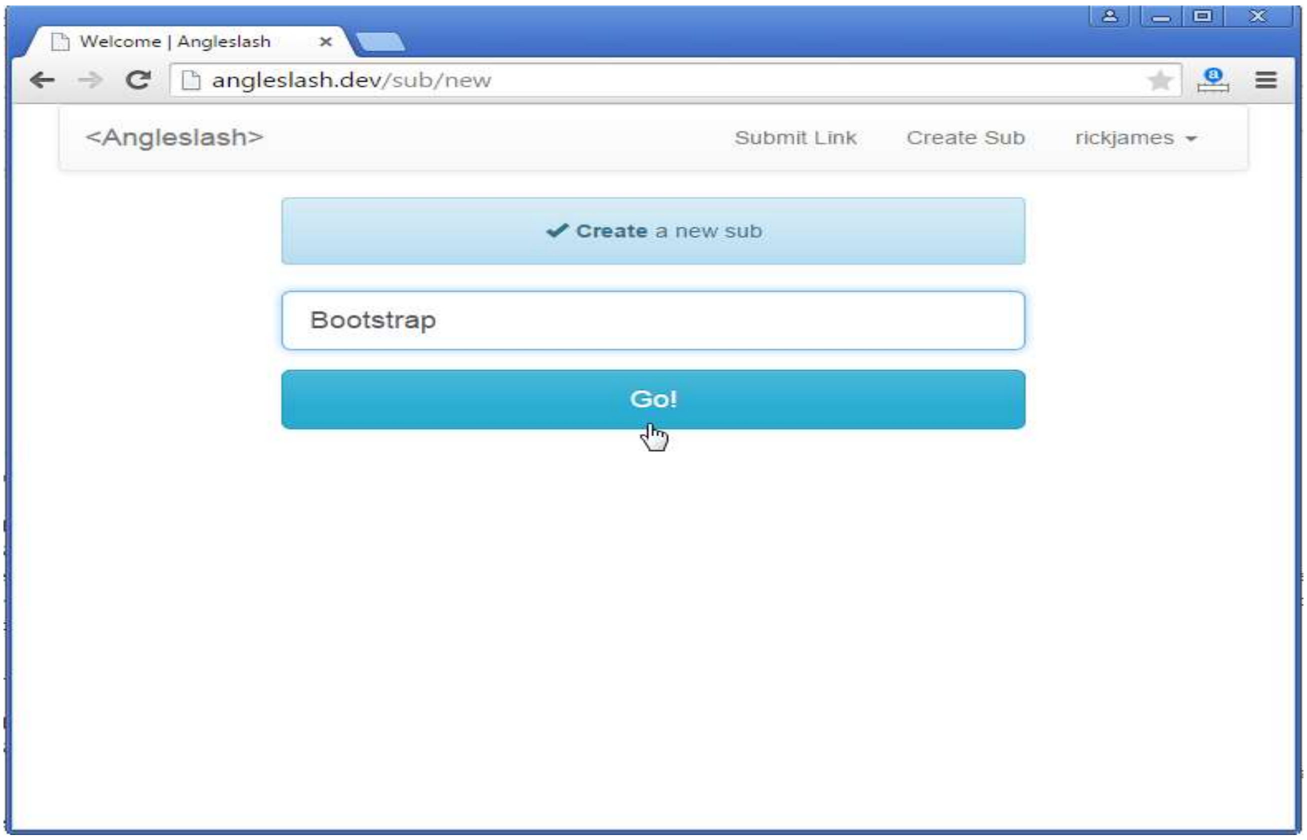
Name: rickjames

E-Mail Address: ricky@thisisaparty.com

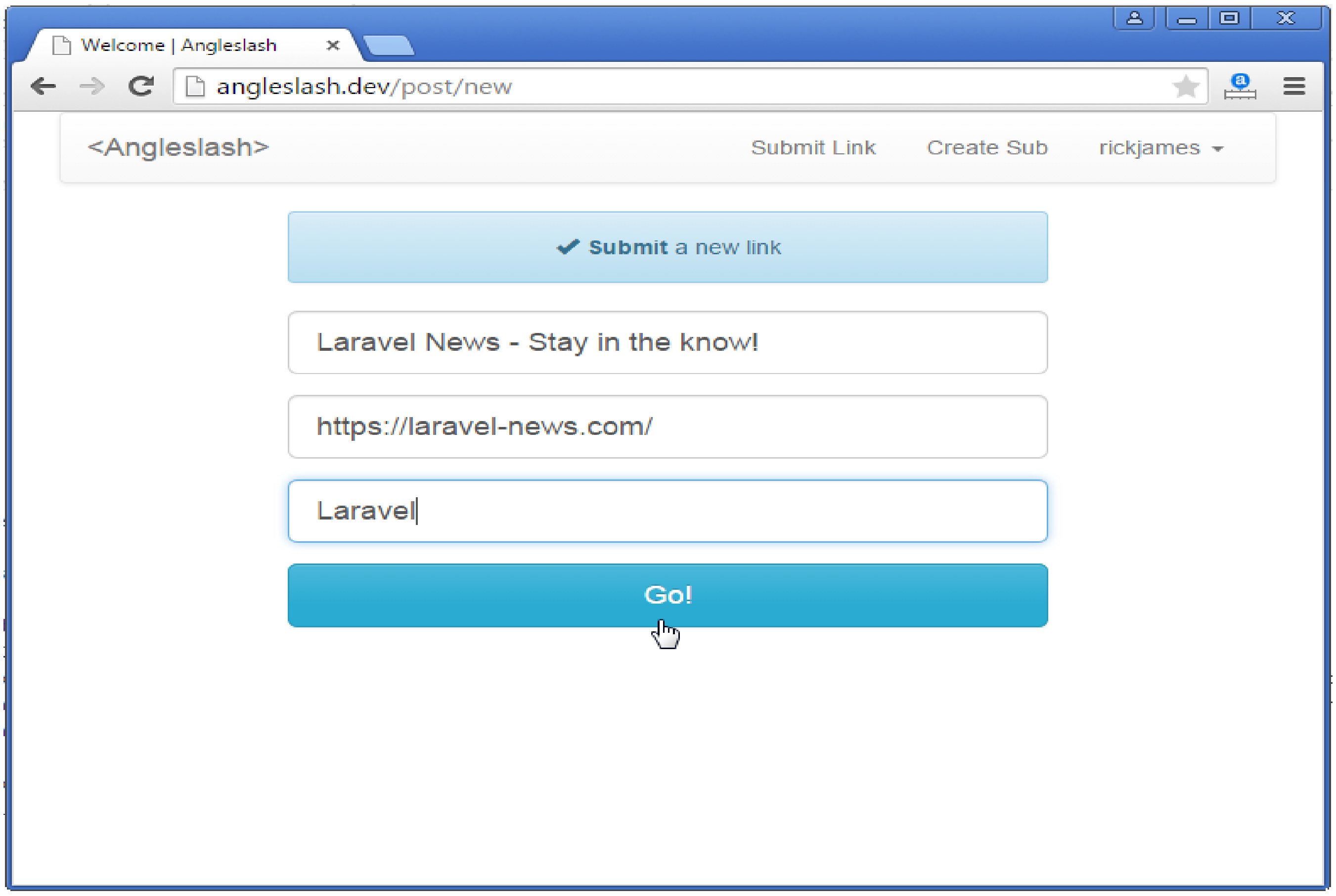
Password: ••••••

Confirm Password: ••••••

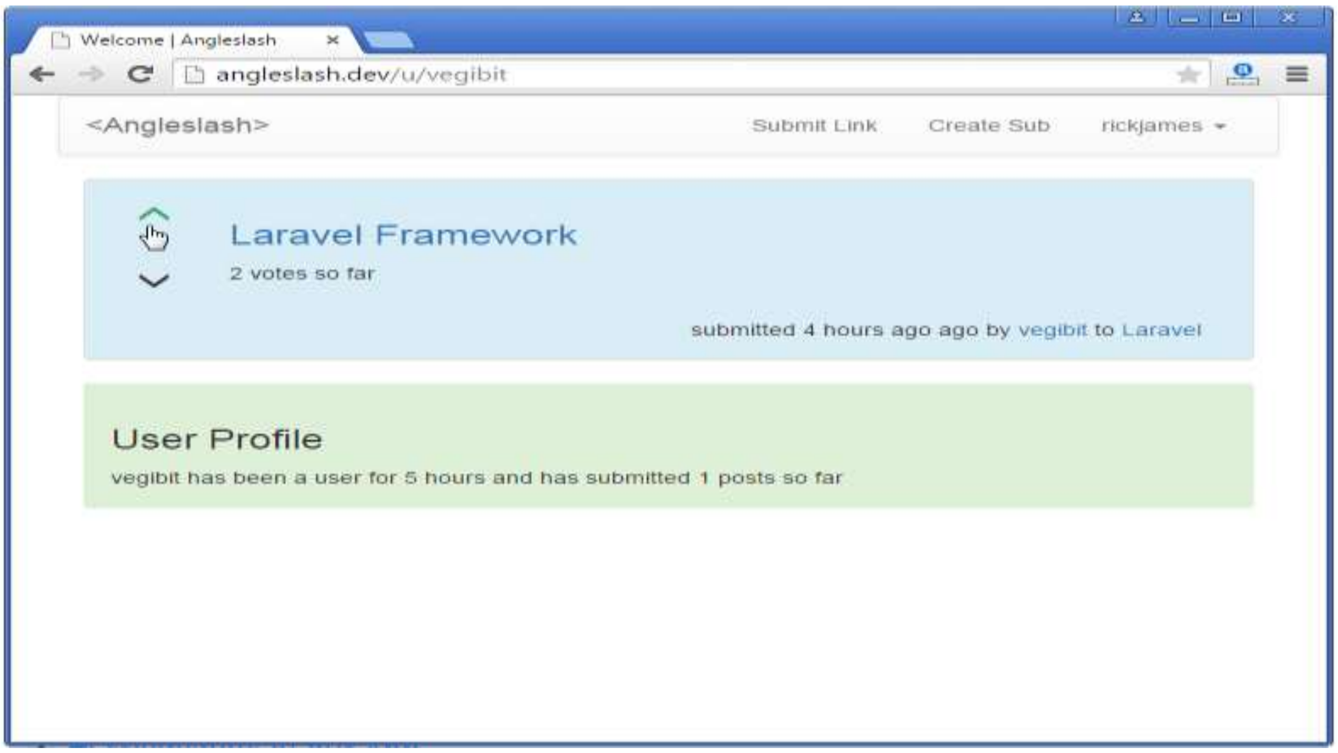
Creating a New Sub



Submitting A New Link



Viewing A User Profile

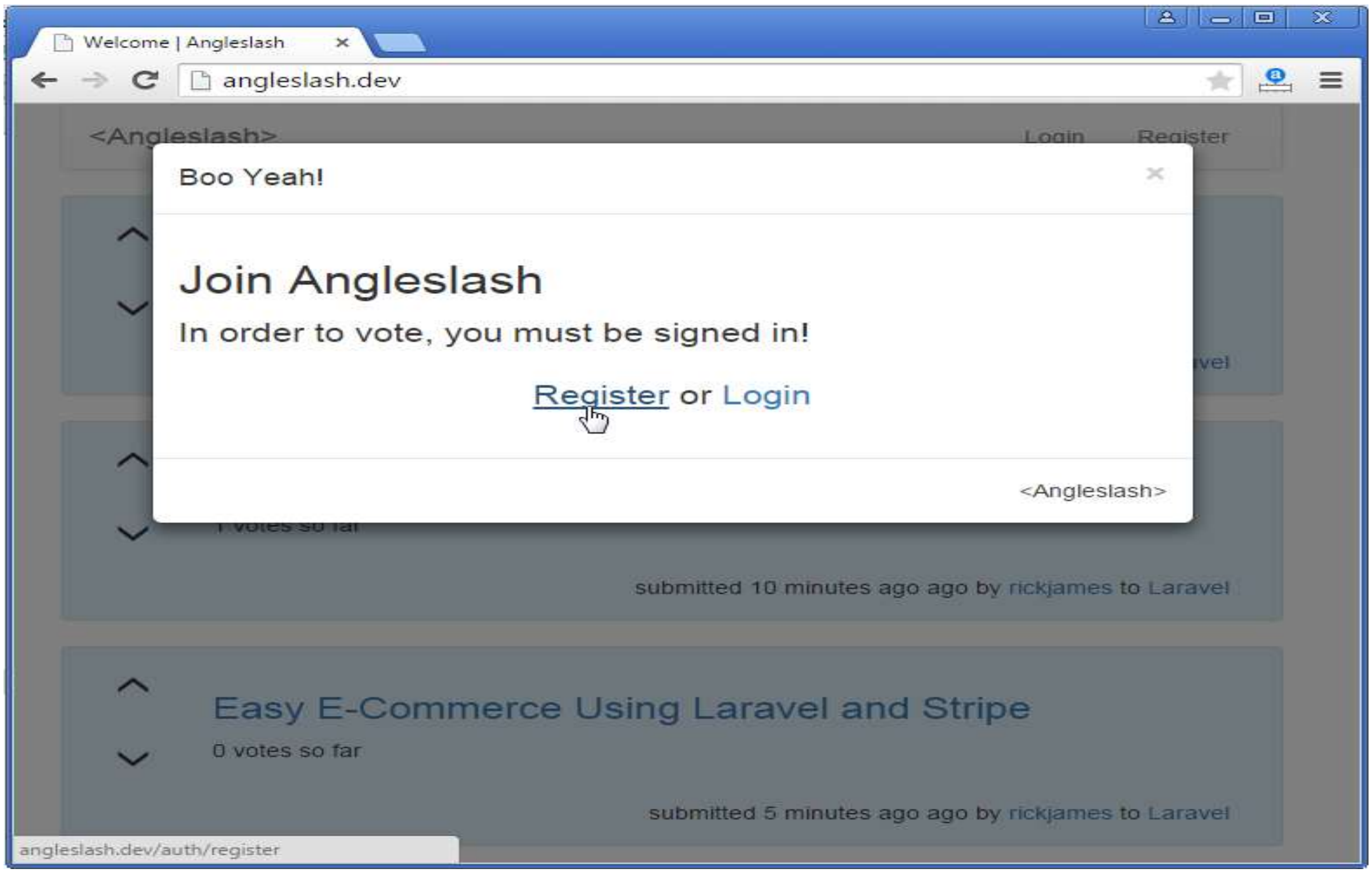


Laravel Framework
2 votes so far
submitted 4 hours ago ago by vegibit to Laravel

User Profile
vegibit has been a user for 5 hours and has submitted 1 posts so far

Cool Modal Effect for Logged Out Users

If a guest or non logged in user tries to click the vote buttons, either up or down, they will trigger a modal that will request them to either register for a new account, or log in with their existing account.



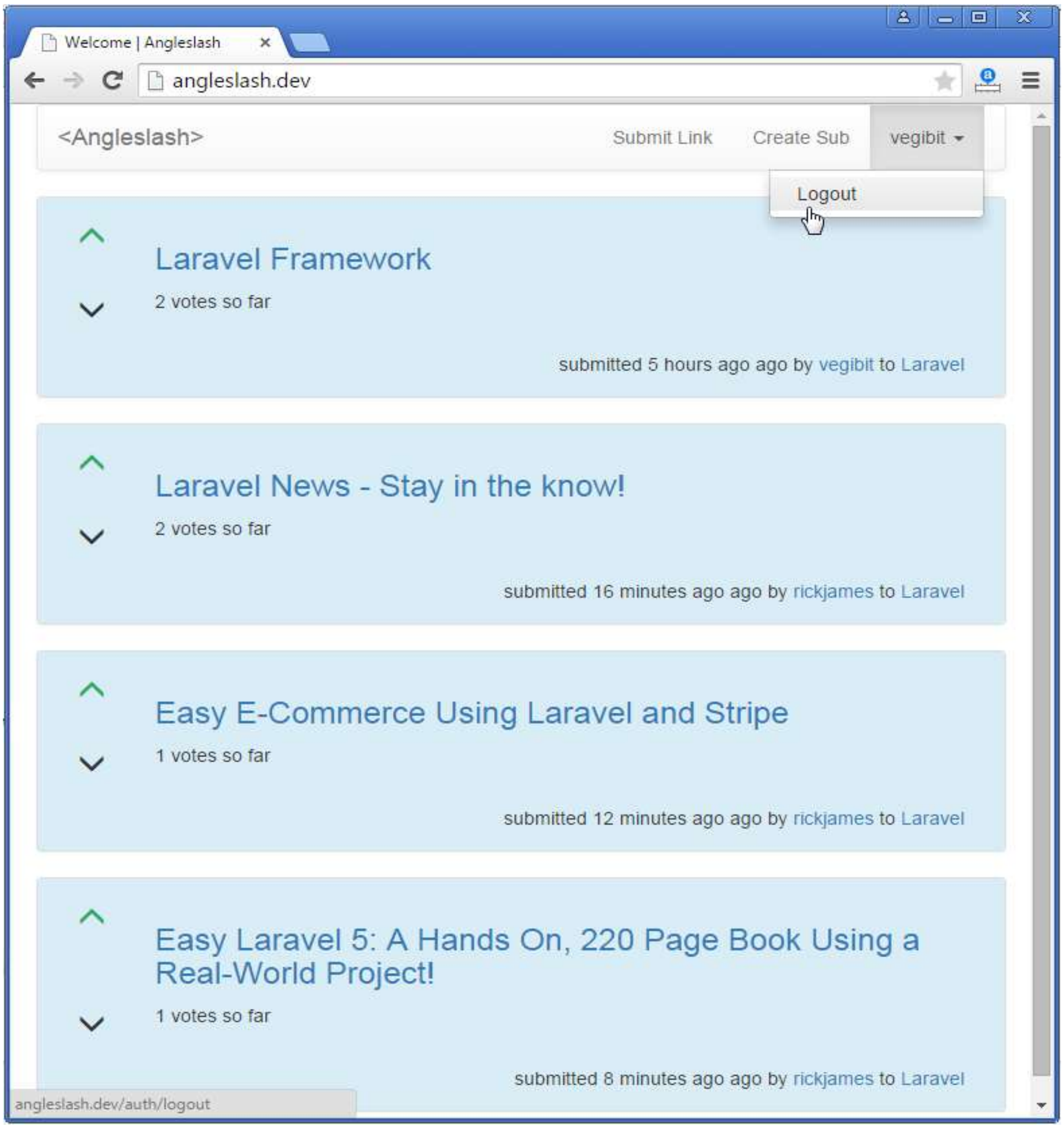
Boo Yeah!

Join Angleslash
In order to vote, you must be signed in!
Register or Login

1 votes so far
submitted 10 minutes ago ago by rickjames to Laravel

Easy E-Commerce Using Laravel and Stripe
0 votes so far
submitted 5 minutes ago ago by rickjames to Laravel

Viewing The Front Page of Angleslash



Laravel Framework
2 votes so far
submitted 5 hours ago ago by vegibit to Laravel

Laravel News - Stay in the know!
2 votes so far
submitted 16 minutes ago ago by rickjames to Laravel

Easy E-Commerce Using Laravel and Stripe
1 votes so far
submitted 12 minutes ago ago by rickjames to Laravel

Easy Laravel 5: A Hands On, 220 Page Book Using a Real-World Project!
1 votes so far
submitted 8 minutes ago ago by rickjames to Laravel

angleslash.dev/auth/logout

Build A Link Sharing Website With Laravel Conclusion

This has been a really fun tutorial that put the rubber to the road on a sample Laravel application. The goal was to cover an example Laravel workflow in a soup to nuts fashion, starting with Homestead, and covering a wide range of topics including choosing a project name, creating a new application with Composer, setting a namespace with Artisan, using PHP Storm for code editing, configuring a base url, configuring environment settings, running a sample PHP Unit test, creating a routes file, using Artisan to generate Controllers, Models, Migrations, and Form Requests, as well as completing very basic styling with Bootstrap.

Reference(s):

1. PHP Tutorials for Beginners at https://www.guru99.com/php-tutorials.html
2. https://www.w3schools.com/php-tutorials.html
3. Steve Suehring, Tim Converse, and Joyce Park, PHP6 and MySQL6 BIBLE
4. Robin Nixon, Learning PHP, MySQL and JavaScript, 4th Edition
5. https://www.laravel.com/docs/7.x/container#:~:text=The%20Laravel%20service%20container%20is,cases%2C%20%22setter%22%20methods.
6. https//www.getintopc.com