Web Vulnerability Research

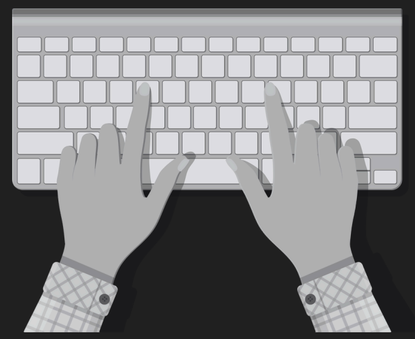
By : Ramadhan Amizudin

MCC-2024

Course Structure

01

Theory



02

Hands-on / Exercise

03

Homework

< Prologue >

WHO AM I

- **Ramadhan Amizudin**

@rempahrz

https://rz.my

Virtual Machine

Credential

Username : student
Password : mcc2024#@!

Exercise Folder

/home/student/classroom

Text Editor

SublimeText

The Usual Command

$ ./bin/start.sh
$ ./bin/stop.sh

Theory

- PHP - The Language
- Laravel Framework

PHP - The Language

1. Application Input
2. Strings
3. Classess and Objects

PHP Fundamentals

PHP is an interpreted language and as such

1. Supports a full object-oriented programming interface
2. Is dynamically typed
3. Does not provide any real memory safety features
4. Does not provide any real support for concurrency
5. Utilizes very little reflection
6. Supports custom serialization

PHP Application Input

$GLOBALS – Global variables defined in the application

$_SERVER – Access client and response data (eg: req/res HEADER)

$_REQUEST – Any **GET/POST** request parameters

$_POST – Any **POST** request parameters

$_GET – Any **GET** request parameters

$_FILES – HTTP **File Upload** variables

$_ENV – Any PHP environment variables

$_COOKIE – Any cookies parameters

$_SESSION – Session variables

PHP Application Input

$_GET / $_POST / $_COOKIE / $_REQUEST

Super global variable to access user-input, unfiltered by default.

$_GET = URL Query

$_POST = POST parameter

$_COOKIE = Cookie

$_REQUEST = $_GET+$_POST order depending on PHP setting “request_order”

and “variables_order”

PHP Application Input

$_FILES - An associative array of item(file) uploaded via HTTP POST.

$_FILES['userfile']['name'] = File name of uploaded file, user controlled, sanitized by basename() by default

$_FILES['userfile']['type'] = The mime type of uploaded file, user controlled, unfiltered by default

$_FILES['userfile']['size'] = The size in bytes

$_FILES['userfile']['tmp_name'] = The temporary file location was stored on the server

$_FILES['userfile']['error'] = The error code associated with this file upload.

$_FILES['userfile']['full_path'] = The file local (user) file path, user controlled, unfiltered by default

PHP Application Input

$_SERVER

Server and execution environment information. Unfiltered by default. HTTP request header is stored in $_SERVER with variable name start with HTTP_, all in capital case, and non-alphanumeric character is replaced with _

Example:

$_SERVER['HTTP_HOST'] = Host header

$_SERVER['HTTP_X_REAL_IP'] = X-Real-IP Header

PHP Strings

A string literal can be specified in four different ways:

1. Single quoted
2. Double quoted
3. Heredoc syntax
4. Nowdoc syntax

Read more:

https://www.php.net/manual/en/language.oop5.inheritance.php

PHP Strings - Single Quoted

The simplest way to specify a string is to enclose it in single quotes

(the character ').

```
echo 'this is a simple string';


echo 'You can also have embedded newlines in
strings this way as it is
okay to do';
```

PHP Strings - Double Quoted

If the string is enclosed in double-quotes ("), PHP will interpret the following escape sequences for special characters.

The most important feature of double-quoted strings is the fact that variable names/function call will be expanded.

PHP Strings - Double Quoted

Escaped characters	
Sequence	**Meaning**
`\n`	linefeed (LF or 0x0A (10) in ASCII)
`\r`	carriage return (CR or 0x0D (13) in ASCII)
`\t`	horizontal tab (HT or 0x09 (9) in ASCII)
`\v`	vertical tab (VT or 0x0B (11) in ASCII)
`\e`	escape (ESC or 0x1B (27) in ASCII)
`\f`	form feed (FF or 0x0C (12) in ASCII)
`\\`	backslash
`\$`	dollar sign
`\"`	double-quote
`\[0-7]{1,3}`	Octal: the sequence of characters matching the regular expression `[0-7]{1,3}` is a character in octal notation (e.g. `"\101" === "A"`), which silently overflows to fit in a byte (e.g. `"\400" === "\000"`)
`\x[0-9A-Fa-f]{1,2}`	Hexadecimal: the sequence of characters matching the regular expression `[0-9A-Fa-f]{1,2}` is a character in hexadecimal notation (e.g. `"\x41" === "A"`)
`\u{[0-9A-Fa-f]+}`	Unicode: the sequence of characters matching the regular expression `[0-9A-Fa-f]+` is a Unicode codepoint, which will be output to the string as that codepoint's UTF-8 representation. The braces are required in the sequence. E.g. `"\u{41}" === "A"`

PHP Strings - Heredoc

A third way to delimit strings is the heredoc syntax: <<<. After this operator, an identifier is provided, then a newline. The string itself follows, and then the same identifier again to close the quotation. Works the same as double quoted strings. It just uses a string as starting and ending marker.

```
$bar = <<<EOT
foo
EOT;
```

PHP Strings - Nowdoc

Nowdocs are to single-quoted strings what heredocs are to double-quoted strings.

A nowdoc is specified similarly to a heredoc, but no String interpolation is done

inside a nowdoc.

```
<?php
echo <<<'EOD'
Example of string spanning multiple lines
using nowdoc syntax. Backslashes are always treated literally,
e.g. \\ and \'.
EOD;
```

PHP Classes and Objects

- Class Visibility
- Inheritance
- Namespaces

PHP Classes and Objects

Classes

- We can encapsulate code into a class, which can contain properties and methods of different visibility.
-

Public visibility

- Anyone can access public properties or methods
-

Private visibility

- Can only accessed by the class that defines the property or method

PHP Classes and Objects

Protected visibility

- Can be accessed by the class that defines the property or method and by inheriting and parent classes.

PHP Classes and Objects

Inheritance

Inheritance is a well-established programming principle, and PHP makes use of this principle in its object model. This principle will affect the way many classes and objects relate to one another.

For example, when extending a class, the subclass inherits all of the public and protected methods, properties and constants from the parent class. Unless a class overrides those methods, they will retain their original functionality.

Read more: https://www.php.net/manual/en/language.oop5.inheritance.php

PHP Classes and Objects

```php
<?php

class Foo {
    public function printItem($string) {
        echo 'Foo: ' . $string . PHP_EOL;
    }

    public function printPHP() {
        echo 'PHP is great.' . PHP_EOL;
    }
}

class Bar extends Foo {
    public function printItem($string) {
        echo 'Bar: ' . $string . PHP_EOL;
    }
}

$foo = new Foo();
$bar = new Bar();
$foo->printItem('baz'); // Output: 'Foo: baz'
$foo->printPHP();       // Output: 'PHP is great'
$bar->printItem('baz'); // Output: 'Bar: baz'
$bar->printPHP();       // Output: 'PHP is great'

?>
```

PHP Classes and Objects

Namespace

What are namespaces? In the broadest definition namespaces are a way of encapsulating items. This can be seen as an abstract concept in many places.

In the PHP world, namespaces are designed to solve two problems that authors of libraries and applications encounter when creating re-usable code elements such as classes or functions.

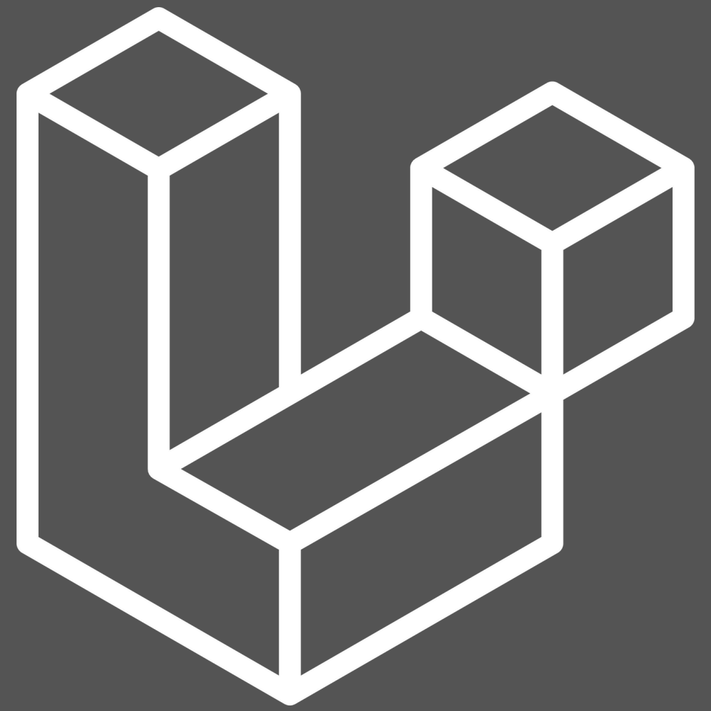
Read more: https://www.php.net/manual/en/language.oop5.inheritance.php

PHP Classes and Objects

Example #1 Namespace syntax example

```php
<?php
namespace my\name; // see "Defining Namespaces" section

class MyClass {}
function myfunction() {}
const MYCONST = 1;

$a = new MyClass;
$c = new \my\name\MyClass; // see "Global Space" section

$a = strlen('hi'); // see "Using namespaces: fallback to global
                   // function/constant" section

$d = namespace\MYCONST; // see "namespace operator and __NAMESPACE__
                        // constant" section
$d = __NAMESPACE__ . '\MYCONST';
echo constant($d); // see "Namespaces and dynamic language features" section
?>
```

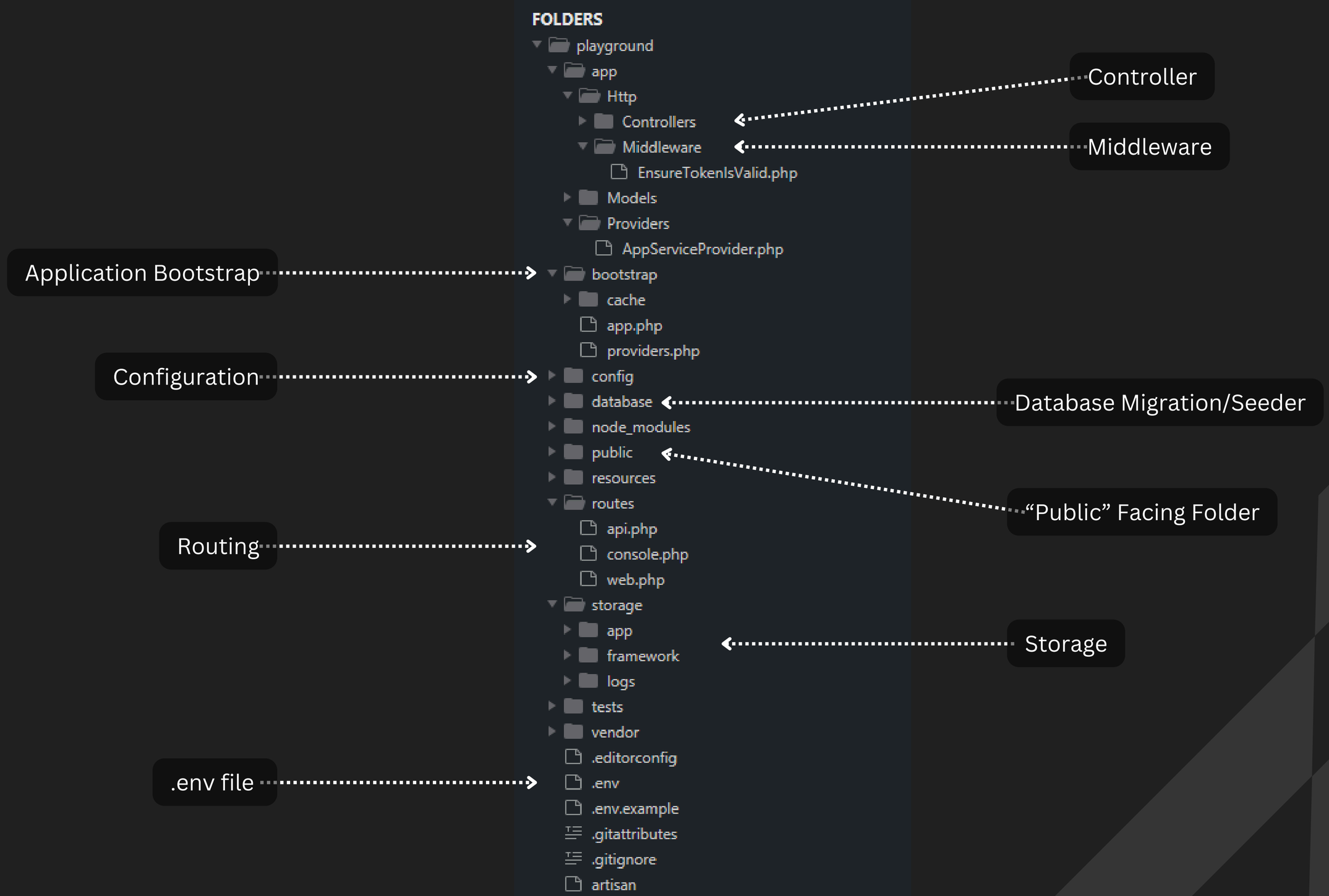
Laravel Framework



Laravel - The Framework

We will speed run learning Laravel framework in 10 minutes

1. Overview
2. Routing
3. Middleware
4. User Input
5. File Upload Handling


FOLDERS
playground
app
Http
Controllers
Middleware
EnsureTokenIsValid.php
Models
Providers
AppServiceProvider.php
bootstrap
cache
app.php
providers.php
config
database
node_modules
public
resources
routes
api.php
console.php
web.php
storage
app
framework
logs
tests
vendor
.editorconfig
.env
.env.example
.gitattributes
.gitignore
artisan
Controller
Middleware
Application Bootstrap
Configuration
Database Migration/Seeder
“Public” Facing Folder
Routing
Storage
.env file

Overview

Controller
Application logic is in this folder

Middleware
Imagine middleware as pre-exec logic checker/another logic. Everything go thru middleware (if defined) before executing any code in Controller

Application Bootstrap
This is the Laravel framework start. If you are lost, this folder is right place to investigate.

Configuration/.env file
This is where application-defined their configuration.

Overview

Database Migration/Seeder
All the application database definition/schema placed in this folder, and seeder is application database default/init value.

“Public” Facing Folder
Web server document rooted pointed to this folder. Everything is this folder is accessible

Storage Folder
All related to cache/session stored in this folder, including logs, etc.

Routing Folder
All routing/navigation definition stored in this folder

Read more: https://laravel.com/docs/11.x/structure#the-root-app-directory

Routing

Multiple way defining routing

Start with routes/web.php - this is default file for apps routing

```
Route::get('/greeting', function () {            <---------------- GET request /greeting will return hello world
    return 'Hello World';
});

Route::get('/', function () {                    <---------------- GET request / will render welcome page, but
    return view('welcome');
})->middleware('isTokenValid');                  <---------------- the request will be checked by isTokenValid middleware

Route::prefix('admin')->group(function () { <---------------- Route group
    Route::get('/users', function () {
        // Matches The "/admin/users" URL
    });
});
```

Routing

Usually the route file look like this.

```
use App\Http\Controllers\UserController;

Route::get('/user', [UserController::class, 'index']);
```

GET request to /user will invoke method **index** in **UserController** class

Location of controller is in app/Http/Controllers/UserController.php

Routing

There are more with routes/api.php. By default in bootstrap/app.php

```
return Application::configure(basePath: dirname(__DIR__))
    ->withRouting(
        web: __DIR__.'/../routes/web.php',
        api: __DIR__.'/../routes/api.php',
        commands: __DIR__.'/../routes/console.php',
        health: '/up',
    )
```

it mean all route in api.php will automatically have prefix /api unless defined otherwise

```
return Application::configure(basePath: dirname(__DIR__))
->withRouting(
    api: __DIR__.'/../routes/api.php',
    apiPrefix: 'api/admin'
```

Routing

Available Router Method

```
Route::get($uri, $callback);
Route::post($uri, $callback);
Route::put($uri, $callback);
Route::patch($uri, $callback);
Route::delete($uri, $callback);
Route::options($uri, $callback);
Route::any($uri, $callback);
Route::match(['get', 'post', $method..], $uri, $callback);
```

Read more: https://laravel.com/docs/11.x/routing

Middleware

There are multiple way to define middleware.

1. Global Middleware
2. Route-based Middleware
3. Controller-based Middleware

Middleware can be set via alias, class or group. All middleware alias is defined in bootstrap/app.php file

Middleware

Global Middleware is defined in bootstrap/app.php file

```
->withMiddleware(function (Middleware $middleware) {
    $middleware->use([
        \Illuminate\Foundation\Http\Middleware\InvokeDeferredCallbacks::class,
        // \Illuminate\Http\Middleware\TrustHosts::class,
        \Illuminate\Http\Middleware\TrustProxies::class,
        \Illuminate\Http\Middleware\HandleCors::class,
        \Illuminate\Foundation\Http\Middleware\PreventRequestsDuringMaintenance::class,
        \Illuminate\Http\Middleware\ValidatePostSize::class,
        \Illuminate\Foundation\Http\Middleware\TrimStrings::class,
        \Illuminate\Foundation\Http\Middleware\ConvertEmptyStringsToNull::class,
    ]);
})
```

Middleware

For route-based middleware it look like this.

```
Route::get('/', function () {
    return view('welcome');
})->middleware('isTokenValid');
```

isTokenValid is middleware alias, open up bootstrap/app.php will reveal which class responsible for handling this

```
->withMiddleware(function (Middleware $middleware) {
    $middleware->alias([
        'isTokenValid' => \App\Http\Middleware\EnsureTokenIsValid::class,
    ]);
})
```

EnsureTokenIsValid class is responsible for handling isTokenValid, and the class file is located in app/Http/Middleware/EnsureTokenIsValid.php

Middleware

Or by using the class name directly

```
Route::get('/', function () {
    return view('welcome');
})->middleware(\App\Http\Middleware\EnsureTokenIsValid::class);
```

EnsureTokenIsValid class is responsible for handling isTokenValid, and the class file is located in app/Http/Middleware/EnsureTokenIsValid.php

Middleware

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;

class EnsureTokenIsValid
{
    public function handle(Request $request, Closure $next): Response
    {
        $token = $request->input('x-token', false);
        if($token != 'mysecret-token') {
            throw new \Exception('Invalid token');
        }
        return $next($request);
    }
}
```

Middleware

In controller middleware

```php
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use Illuminate\Routing\Controllers\HasMiddleware;
use Illuminate\Routing\Controllers\Middleware;

class UserController extends Controller implements HasMiddleware
{
    /**
     * Get the middleware that should be assigned to the controller.
     */
    public static function middleware(): array
    {
        return [
            'auth',
            new Middleware('log', only: ['index']),
            new Middleware('subscribed', except: ['store']),
        ];
    }

    // ...
}
```

```php
use Closure;
use Illuminate\Http\Request;

/**
 * Get the middleware that should be assigned to the controller.
 */
public static function middleware(): array
{
    return [
        function (Request $request, Closure $next) {
            return $next($request);
        },
    ];
}
```

Different between Version

For Laravel <= 10
Middleware is defined in file: app/Http/Kernel.php

User Input

The most basic of all is

```
$input = $request->all();
```

All request will be assign to $input regardless of request method

```
GET /?var1=b or
POST /
var1=b
```

access via $input[‘var1']

```
$name = $request->input('name'); <-- will take user input from GET or POST
$name = $request->input('name', ‘sally’); <-- same but with default value
```

User Input

```
$input = $request->query(‘name’); <-- Only from query string
$query = $request->query(); <-- take all from query string
```

Accessing data from json.

```
$name = $request->input('user.name');
Request: {”user”:{”name”:”aaa”}}
```

From header

```
$value = $request->header('X-Header-Name');
```

File Upload Handling

Accessing getting file upload.

```
$photo = $request->file('photo') <-- file upload via parameter photo
```

Getting file extension of uploaded file

```
$server_side_extension = $photo->extension();
$client_side_extension = $photo->getClientOriginalExtension();
```

Storing uploaded file.. there tons of way!

```
->store($foldername)
$path = Storage::putFile('photo', $request->file('photo'));
```

Read more:
https://laravel.com/docs/11.x/filesystem
https://laravel.com/docs/11.x/requests#files