

MODULE 5

JSON Data Interchange Format: Syntax, Data Types, Object, JSON Schema, Manipulating JSON data with PHP Web Development Frameworks: Laravel Overview-Features of Laravel-Setting up a Laravel Development Environment-Application structure of Laravel-Routing -Middleware-Controllers- Route Model Binding-Views-Redirections-Request and Responses.

JSON

- JavaScript Object Notation
 - lightweight format for storing and transporting data
 - often used when data is sent from a server to a web page
 - "self-describing" and easy to understand
-

JSON

- syntactically identical to the code for creating JavaScript objects
- JavaScript program can easily convert JSON data into native JavaScript objects
- JSON format is text only
- Code for reading and generating JSON data can be written in any programming language

EXAMPLE

```
{"employees": [  
    {"name": "Sonoo", "email": "sonoojaiswal1987@gmail.com"},  
    {"name": "Rahul", "email": "rahul32@gmail.com"},  
    {"name": "John", "email": "john32bob@gmail.com"}  
]
```

- JSON syntax is basically considered as a subset of JavaScript syntax;
- Data is represented in name/value pairs.
- Curly braces hold objects and each name is followed by ':'(colon), the name/value pairs are separated by , (comma).
- Square brackets hold arrays and values are separated by ,(comma).

JSON Data - A Name and a Value

JSON data is written as name/value pairs, just like JavaScript object properties.

A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:

```
"firstName": "John"
```

JSON names require double quotes. JavaScript names do not.

JSON Objects

JSON objects are written inside curly braces.

Just like in JavaScript, objects can contain multiple name/value pairs:

```
{"firstName": "John", "lastName": "Doe"}
```

JSON Arrays

JSON arrays are written inside square brackets.

Just like in JavaScript, an array can contain objects:

```
"employees": [
    {"firstName": "John", "lastName": "Doe"},
    {"firstName": "Anna", "lastName": "Smith"},
    {"firstName": "Peter", "lastName": "Jones"}
]
```

In the example above, the object "employees" is an array. It contains three objects.

Each object is a record of a person (with a first name and a last name).

JSON format supports the following data types

- **Number**

Eg: {"roll": 25}

- **String**

Eg: {"name": "minnu"}

- **Boolean**

Eg: {"name": true}

- **Array**

```
{  
  "employees": [ "John", "Anna", "Peter" ]  
}
```

- **Object**

```
{  
  "student": { "name": "mia", "age": 30, "place": "kollam" }  
}
```

- **Null**

```
{ "firstname": null }
```

- The \$schema keyword states that this schema is written according to a specific draft of the standard and used for a variety of reasons, primarily version control.
- The \$id keyword defines a URI for the schema, and the base URI that other URI references within the schema are resolved against.
- The title and description annotation keywords are descriptive only. They do not add constraints to the data being validated. The intent of the schema is stated with these two keywords.
- The type validation keyword defines the first constraint on our JSON data and in this case it has to be a JSON Object.

- { "\$schema": "https://json-schema.org/draft/2020-12/schema",
"\$id": "https://example.com/product.schema.json",
"title": "Product",
"description": "A product in the catalog",
"type": "object" }

JSON Schema

- The name for this declaration will always be “\$schema,” and the value will always be the link for the draft version

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#"  
}
```

- The second name-value pair in our JSON Schema Document will be the title
- *Format for a document that represents a cat*

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "title": "Cat"  
}
```

- In the third name-value pair of our JSON Schema Document, we will define the properties that we want to be included in the JSON. The property value is essentially a skeleton of the name-value pairs of the JSON we want. Instead of a literal value, we have an object that defines the data type, and optionally the description

Defining the properties for a cat

```
{  
  "$schema": "http://json-schema.org/draft-04/schema#",  
  "title": "Cat",  
  "properties": {  
    "name": {  
      "type": "string"  
    },  
    "age": {  
      "type": "number",  
      "description": "Your cat's age in years."  
    },  
    "declawed": {  
      "type": "boolean"  
    }  
  }  
}
```

This JSON conforms to our JSON Schema for “Cat”

```
{  
  "name": "Fluffy",  
  "age": 2,  
  "declawed": false  
}
```

```
{  
    "$schema": "http://json-schema.org/draft-04/schema#",  
    "title": "Cat",  
    "properties": {  
        "name": {  
            "type": "string"  
        },  
        "age": {  
            "type": "number",  
            "description": "Your cat's age in years."  
        },  
        "declawed": {  
            "type": "boolean"  
        },  
        "description": {  
            "type": "string"  
        }  
    },  
    "required": [  
        "name",  
        "age",  
        "declawed"  
    ]  
}
```

Valid JSON

```
{  
    "name": "Fluffy",  
    "age": 2,  
    "declawed": false,  
    "description": "Fluffy loves to sleep all day."  
}
```

PHP and JSON

- PHP has some built-in functions to handle JSON.
- First, we will look at the following two functions:
 - (i) json_encode()
 - (ii) json_decode()

PHP - json_encode()

- The json_encode() function is used to encode a value to JSON format.

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

<?php
$age = array("Peter"=>35, "Ben"=>37, "Joe"=>43);

echo json_encode($age);
?>

</body>
</html>
```

Output

{"Peter":35, "Ben":37, "Joe":43}

This example shows how to encode an indexed array into a JSON array:

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

<?php
$cars = array("Volvo", "BMW", "Toyota");

echo json_encode($cars);
?>

</body>
</html>
```

Output

["Volvo", "BMW", "Toyota"]

PHP - json_decode()

- The `json_decode()` function is used to decode a JSON object into a PHP object or an associative array.

Example

- This example decodes JSON data into a PHP object:

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

<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

var_dump(json_decode($jsonobj));
?>

</body>
</html>
```

Output

object(stdClass)#1 (3) { ["Peter"]=> int(35) ["Ben"]=> int(37)
["Joe"]=> int(43) }

- The `json_decode()` function returns an object. The `json_decode()` function has a second parameter, and when set to true, JSON objects are decoded into associative arrays.
- <!DOCTYPE html>
<html>
<body>

```
<?php  
$jsonobj = '{ "Peter":35,"Ben":37,"Joe":43}';  
var_dump(json_decode($jsonobj, true));  
?>  
</body>  
</html>
```

Output

- array(3) { ["Peter"]=> int(35) ["Ben"]=> int(37) ["Joe"]=> int(43) }

PHP - Accessing the Decoded Values

- <!DOCTYPE html>
<html>
<body>

<?php
\$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

\$obj = json_decode(\$jsonobj);

echo \$obj->Peter;
echo \$obj->Ben;
echo \$obj->Joe;
?>

</body>
</html>

Output

353743

Example

This example shows how to access the values from a PHP associative array:

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

<?php
$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

$arr = json_decode($jsonobj, true);

echo $arr["Peter"];
echo $arr["Ben"];
echo $arr["Joe"];
?>

</body>
</html>
```

Output

353743

PHP - Looping Through the Values

- <!DOCTYPE html>
<html>
<body>

<?php
\$jsonobj = '{"Peter":35,"Ben":37,"Joe":43}';

\$obj = json_decode(\$jsonobj);

foreach(\$obj as \$key => \$value) {
 echo \$key . " => " . \$value . "
";
}
?>

</body>
</html>

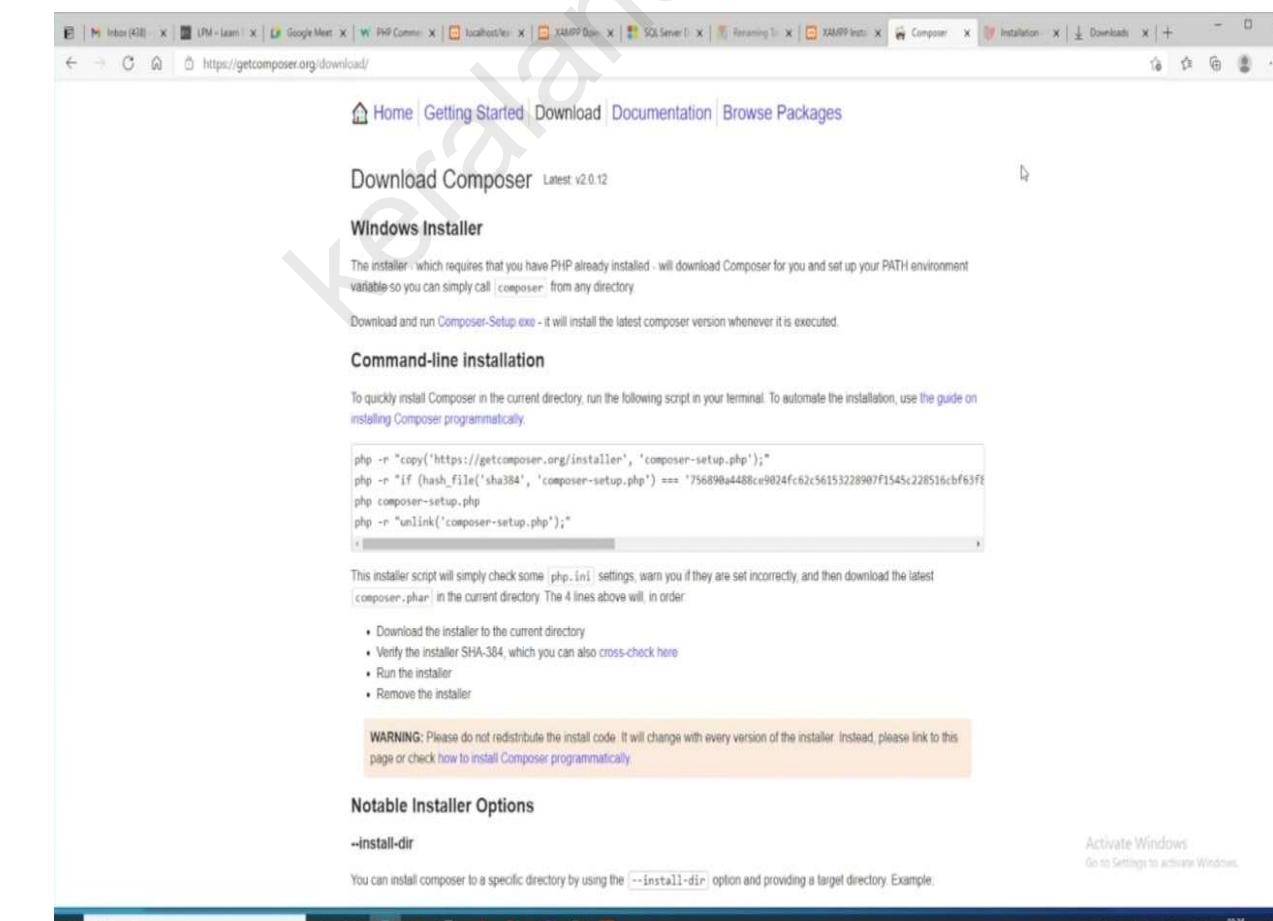
Output

Peter => 35
Ben => 37
Joe => 43

Laravel installation



The screenshot shows the Apache Friends XAMPP download page. At the top, there's a navigation bar with links for Apache Friends, Download, Add-ons, Hosting, Community, and About. A search bar and language selection (EN) are also present. The main heading is "XAMPP Apache + MariaDB + PHP + Perl". Below it, a section titled "What is XAMPP?" describes it as the most popular PHP development environment. To the right is a large orange square icon with a white play button symbol and the word "XAMPP" below it. Below the icon are four download links: "Download Click here for other versions" (green arrow), "XAMPP for Windows 8.0.3 (PHP 8.0.3)" (grey), "XAMPP for Linux 8.0.3 (PHP 8.0.3)" (grey), and "XAMPP for OS X 8.0.3 (PHP 8.0.3)" (grey). A banner at the bottom left announces "New XAMPP release 7.3.27, 7.4.16". A note from Apache Friends says they've released a new version of XAMPP, with download links and a "Read more >" link. At the bottom, there are links for "About Apache Friends", "Community", "Recent Discussions", and "Activate Windows".



The screenshot shows the getcomposer.org/download/ page. The top navigation bar includes links for Home, Getting Started, Download, Documentation, and Browse Packages. The main heading is "Download Composer Latest: v2.0.12". Below it, there's a section for "Windows Installer" which explains the installer requires PHP and provides instructions to run "composer-setup.exe". It also lists steps for "Command-line installation" using the terminal command:

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
php -r "if (hash_file('sha384', 'composer-setup.php') === '756890a448ce9024fc62c5615328907f1545c228516cbf63f7'
php composer-setup.php
php -r "unlink('composer-setup.php');"
```

A note states that the installer will check PHP settings and download the latest "composer.phar" file. Below this, a "WARNING" box cautions against redistributing the install code. The page also lists "Notable Installer Options" like `-install-dir` and provides an example of using the option to install to a specific directory.

<https://laravel.com/docs/8.x/installation#getting-started-on-windows>

curl -s "https://laravel.build/example-app-withmysql,redis" | bash

If you do not specify which services you would like configured, a default stack of mysql, redis, Elasticsearch, Elasticsearch, and selenium will be configured.

Installation Via Composer

If your computer already has PHP and Composer installed, you may create a new Laravel project by using Composer directly. After the application has been created, you may start Laravel's local development server using the Artisan CLI's serve command:

```
composer create-project laravel/laravel example-app
cd example-app
php artisan serve
```

☰ The Laravel Installer

Or, you may install the Laravel Installer as a global Composer dependency:

```
composer global require laravel/installer
laravel new example-app
cd example-app
php artisan serve
```



Deploy Laravel and PHP applications on
DigitalOcean, Linode,
AWS.
Activate Windows
Go to Settings to activate Windows.
Add to GitHub

Type here to search

22:06 22-04-2021

<https://laravel.com/docs/8.x/installation#getting-started-on-windows>

curl -s "https://laravel.build/example-app-withmysql,redis" | bash

If you do not specify which services you would like configured in your new application's docker-compose.yml file, Available services include mysql, pgsql, mariadb, redis, memcached, Elasticsearch, Elasticsearch, and selenium:

Installation Via Composer

If your computer already has PHP and Composer installed, you may create a new Laravel project by using Composer directly. After the application has been created, you may start Laravel's local development server using the Artisan CLI's serve command:

```
composer create-project laravel/laravel example-app
cd example-app
php artisan serve
```

XAMPP Control Panel v3.2.4 [Compiled: Jun 9th 2019]

Module	Service	Port(s)	Actions
Apache		22:24:43 [Apache] 80, 443	Stop Admin Config Logs Shell
MySQL		22:24:43 [mysqld] 3306	Stop Admin Config Logs Explorer Services
FileZilla		22:24:44 [FileZilla]	Start Admin Config Logs
Mercury		22:24:45 [Mercury]	Start Admin Config Logs
Memcached		22:24:46 [memcached]	Start Admin Config Logs

Attempting to stop Apache (PID: 1844)
Status change detected: stopped
Attempting to stop MySQL app...
Status change detected: stopped
Attempting to start Apache app...
Status change detected: running
Attempting to start MySQL app...
Status change detected: running

composer global require laravel/installer
laravel new example-app
cd example-app
php artisan serve



Deploy Laravel and
PHP applications on
DigitalOcean, Linode,
AWS.
Activate Windows
Go to Settings to activate Windows.
Add to GitHub

Setting environment for using XAMPP for Windows.
krishnadas@DESKTOP-E8007JI c:\xampp
#

```

composer global require laravel/installer

laravel new example-app

cd example-app

php artisan serve

```

Type here to search

FORGE

Activate Windows

Deploy Laravel and PHP applications on DigitalOcean, Linode, & AWS.

Go to Settings to activate Windows.

ADS via CARMON

Setting environment for using XAMPP for Windows.
krishnadas@DESKTOP-E8007JI c:\xampp
php --version
PHP 8.0.3 (cli) (built: Mar 2 2021 23:34:05) (ZTS Visual C++ 2019 x64)
Copyright (c) The PHP Group
Zend Engine v4.0.3, Copyright (c) Zend Technologies

Installation Via Composer

If your computer already has PHP and Composer installed, you can create a Laravel project by using Composer directly. If you have not created a Laravel project yet, you may start Laravel's local development server command:

```

curl -s "https://laravel.build/example-app"

```

curl -s "https://laravel.build/example-app" | composer create-project laravel/laravel example-app --prefer-dist

cd example-app

php artisan serve

The Laravel Installer

Or, you may install the Laravel Installer as a global Composer package:

```

composer global require laravel/installer

laravel new example-app

cd example-app

php artisan serve

```

Activate Windows

Deploy Laravel and PHP applications on DigitalOcean, Linode, & AWS.

Go to Settings to activate Windows.

ADS via CARMON

curl -s "https://laravel.build/example-app" | composer create-project laravel/laravel example-app

Setting environment for using XAMPP for Windows.
krishnadas@DESKTOP-E0007JI c:\xampp
php --version
PHP 8.0.3 (cli) (built: Mar 2 2021 23:34:05) (ZTS Visual C++ 2019 x64)
Copyright (c) The PHP Group
Zend Engine v4.0.3, Copyright (c) Zend Technologies

If your computer already has PHP and Composer installed, you can skip this step. To create a Laravel project by using Composer directly, you may start Laravel's local development server command:

krishnadas@DESKTOP-E0007JI c:\xampp\htdocs
composer create-project laravel/laravel example-app
Creating a "laravel/laravel" project at "./example-app"

composer create-project laravel/laravel example-app
cd example-app
php artisan serve

The Laravel Installer

Or, you may install the Laravel Installer as a global package:

composer global require laravel/installer

laravel new example-app

cd example-app

php artisan serve

Activate Windows
DigitalOcean, Linode, & AWS
Go to Settings to activate Windows.
Add via CAFEROW

curl -s "https://laravel.build/example-app" | composer create-project laravel/laravel example-app

Setting environment for using XAMPP for Windows.
krishnadas@DESKTOP-E0007JI c:\xampp\htdocs
php --version
PHP 8.0.3 (cli) (built: Mar 2 2021 23:34:05) (ZTS Visual C++ 2019 x64)
Copyright (c) The PHP Group
Zend Engine v4.0.3, Copyright (c) Zend Technologies

- Installing sebastian/code-unit-reverse-lookup (2.0.3): Extracting archive
- Installing phpunit/php-code-coverage (9.2.6): Extracting archive
- Installing doctrine/instantiator (1.4.0): Extracting archive
- Installing phpspec/prophecy (1.13.0): Extracting archive
- Installing phar-io/version (3.1.0): Extracting archive
- Installing phar-io/manifest (2.0.1): Extracting archive
- Installing myclabs/deep-copy (1.10.2): Extracting archive
- Installing phpunit/phpunit (9.5.4): Extracting archive
80 package suggestions were added by new dependencies, use 'composer suggest' to see details.
Generating optimized autoload files
> Illuminate\Foundation\ComposerScripts::postAutoloadDump
> @php artisan package:discover --ansi
Discovered Package: facade/ignition
Discovered Package: fideloper/proxy
Discovered Package: fruitcake/laravel-cors
Discovered Package: laravel/sail
Discovered Package: laravel/tinker
Discovered Package: nesbot/carbon
Discovered Package: nunomaduro/collision
Package manifest generated successfully.
74 packages you are using are looking for funding.
Use the 'composer fund' command to find out more!
> @php artisan key:generate --ansi
Application key set successfully.

krishnadas@DESKTOP-E0007JI c:\xampp\htdocs
cd example-app

krishnadas@DESKTOP-E0007JI c:\xampp\htdocs\example-app
#

laravel new example-app

cd example-app

php artisan serve

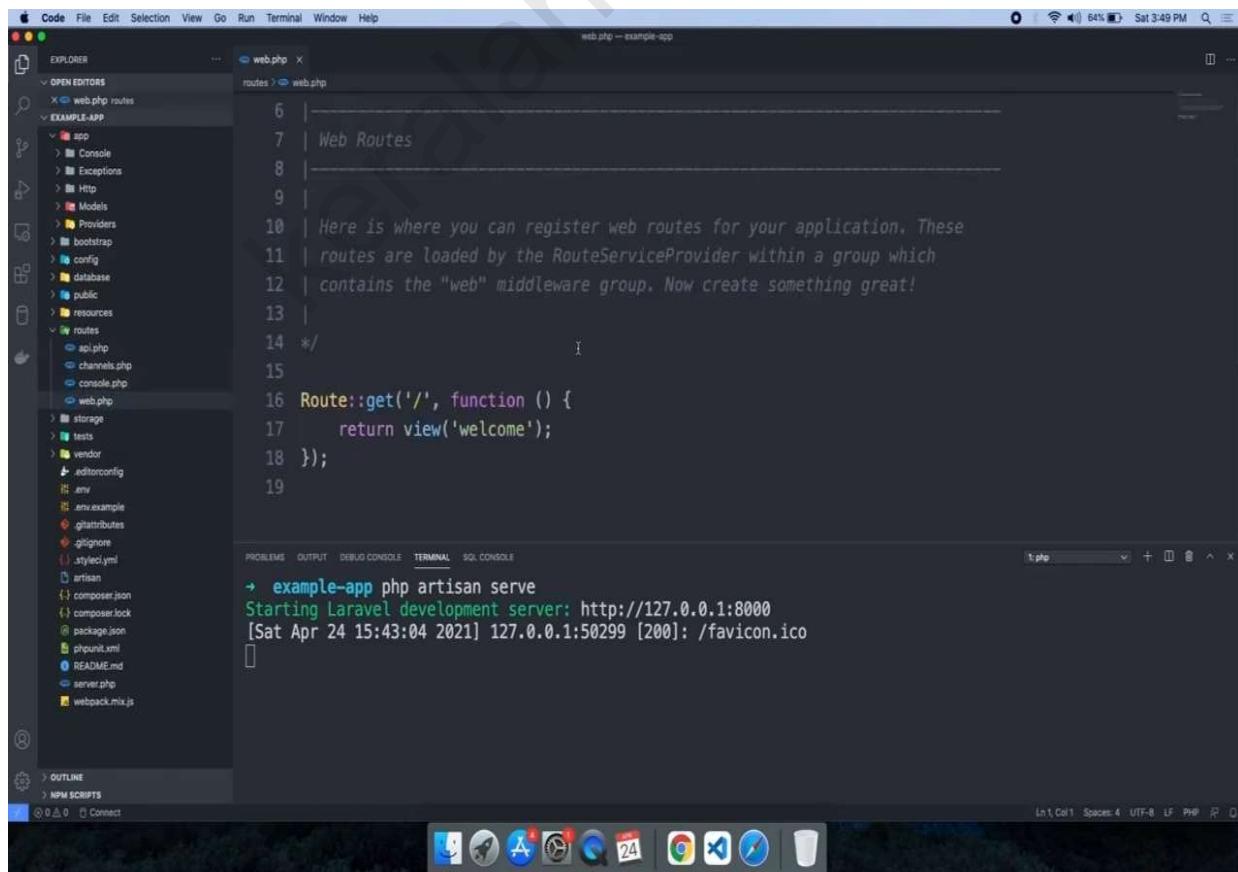
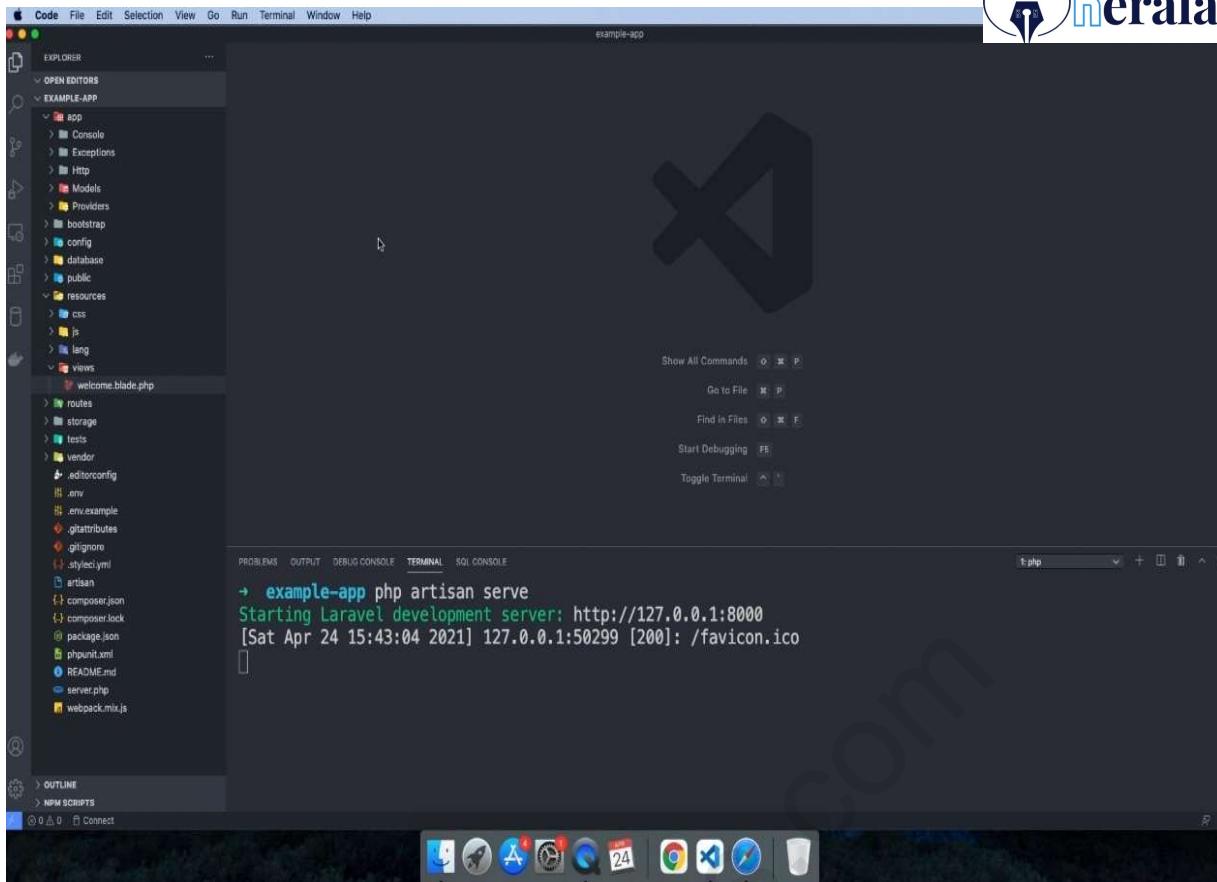
Activate Windows
DigitalOcean, Linode, & AWS
Go to Settings to activate Windows.
Add via CAFEROW

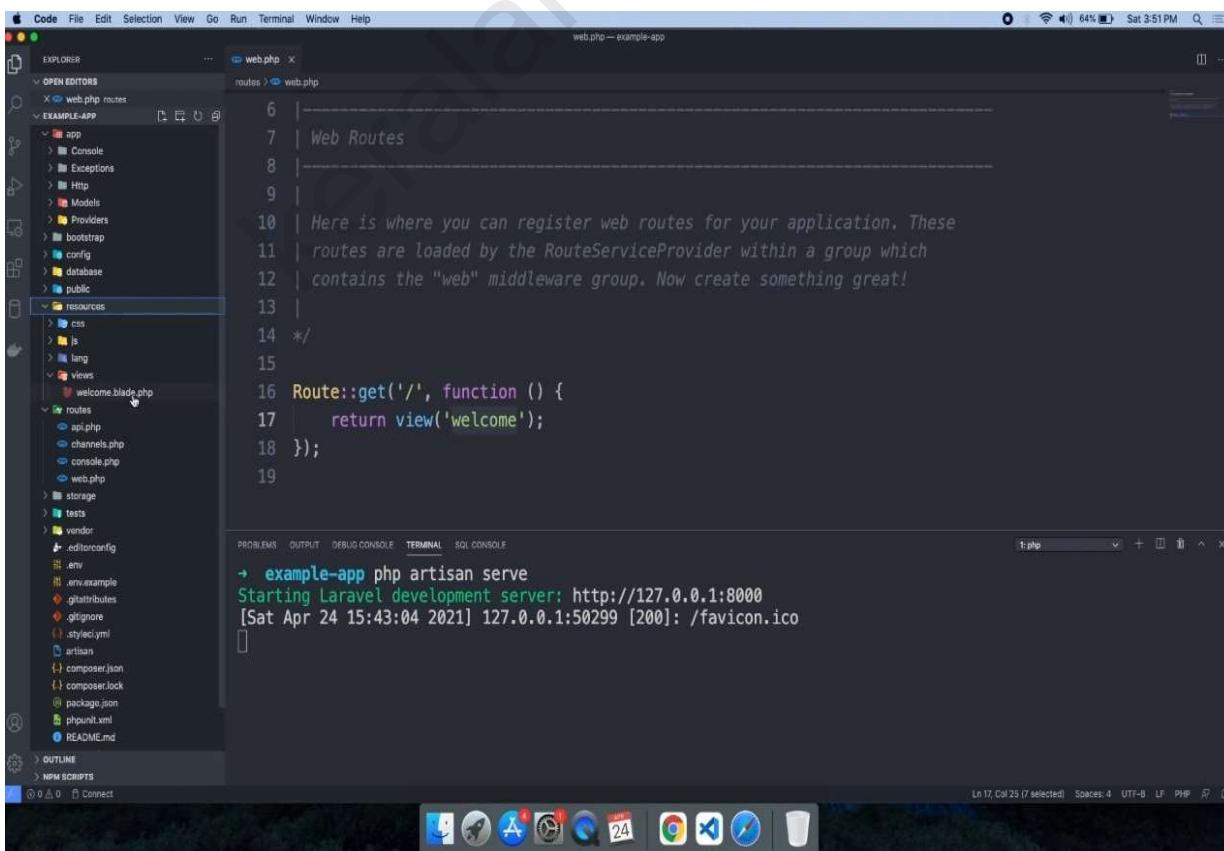
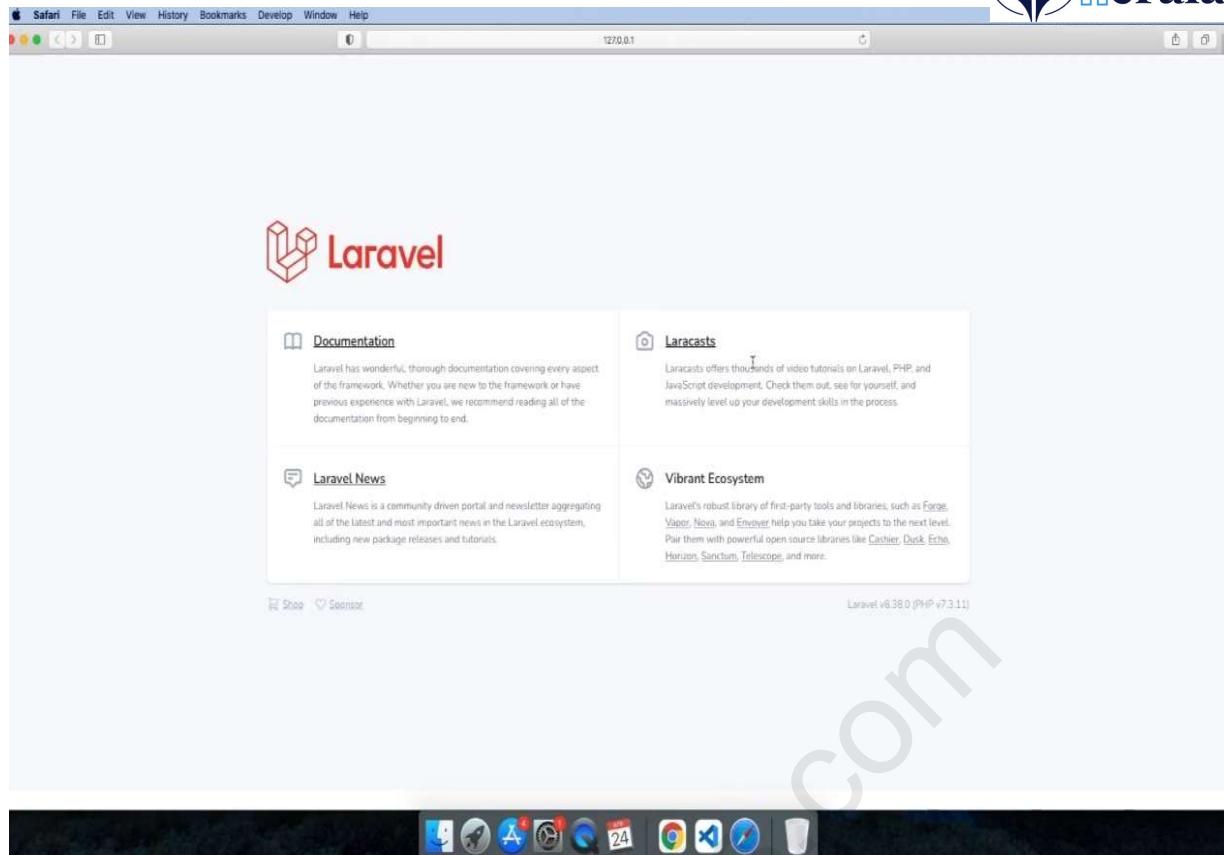
Features of Laravel

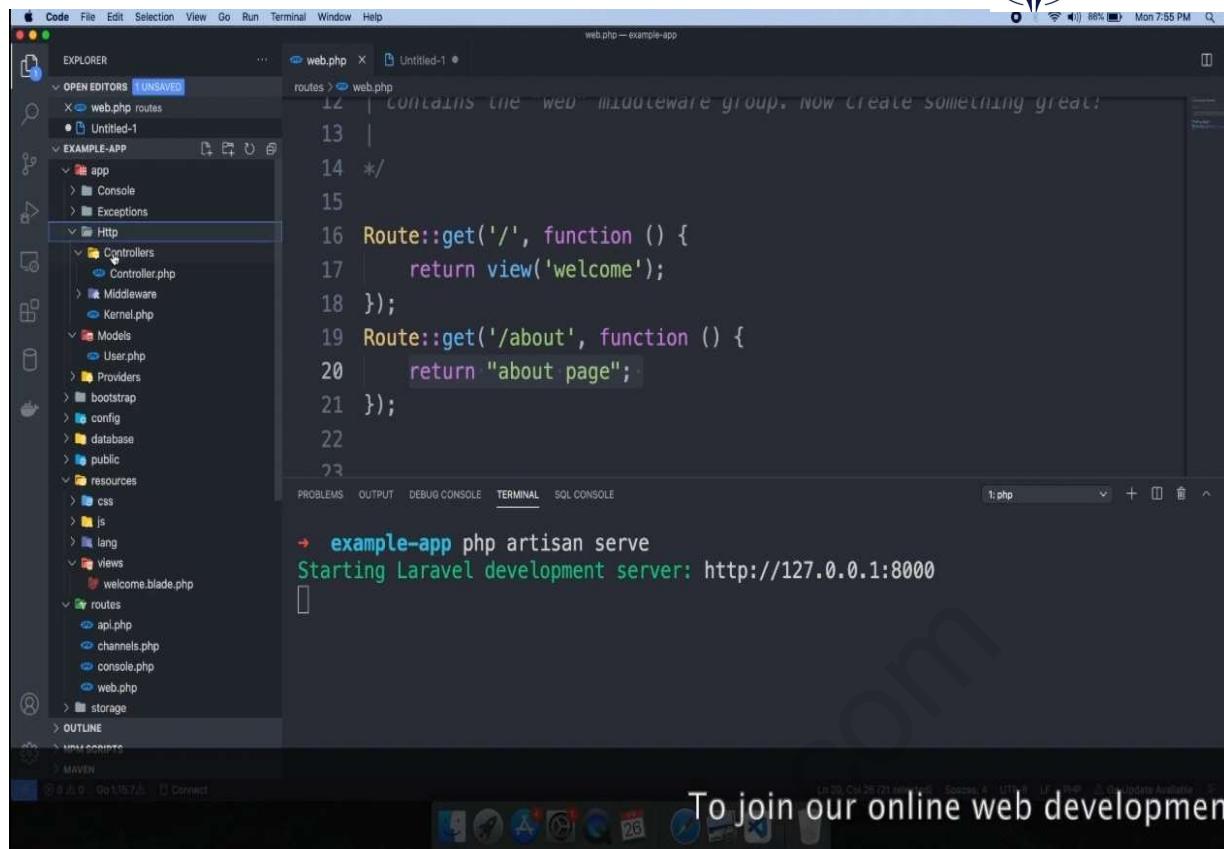
- Template Engine (blade)
- MVC Architecture
- Eloquent ORM (Object Relational Mapping)
- Security
- Artisan
- Libraries & Modular
- Database Migration System
- Unit-Testing

Prerequisite for learning laravel

- Good knowledge of PHP
- Intermediate knowledge of SQL such as MySQL
- Basic knowledge of HTML, CSS & Javascript
- Familiar with command line/linux is also helpful
- For working in real projects learning Git is also mandatory



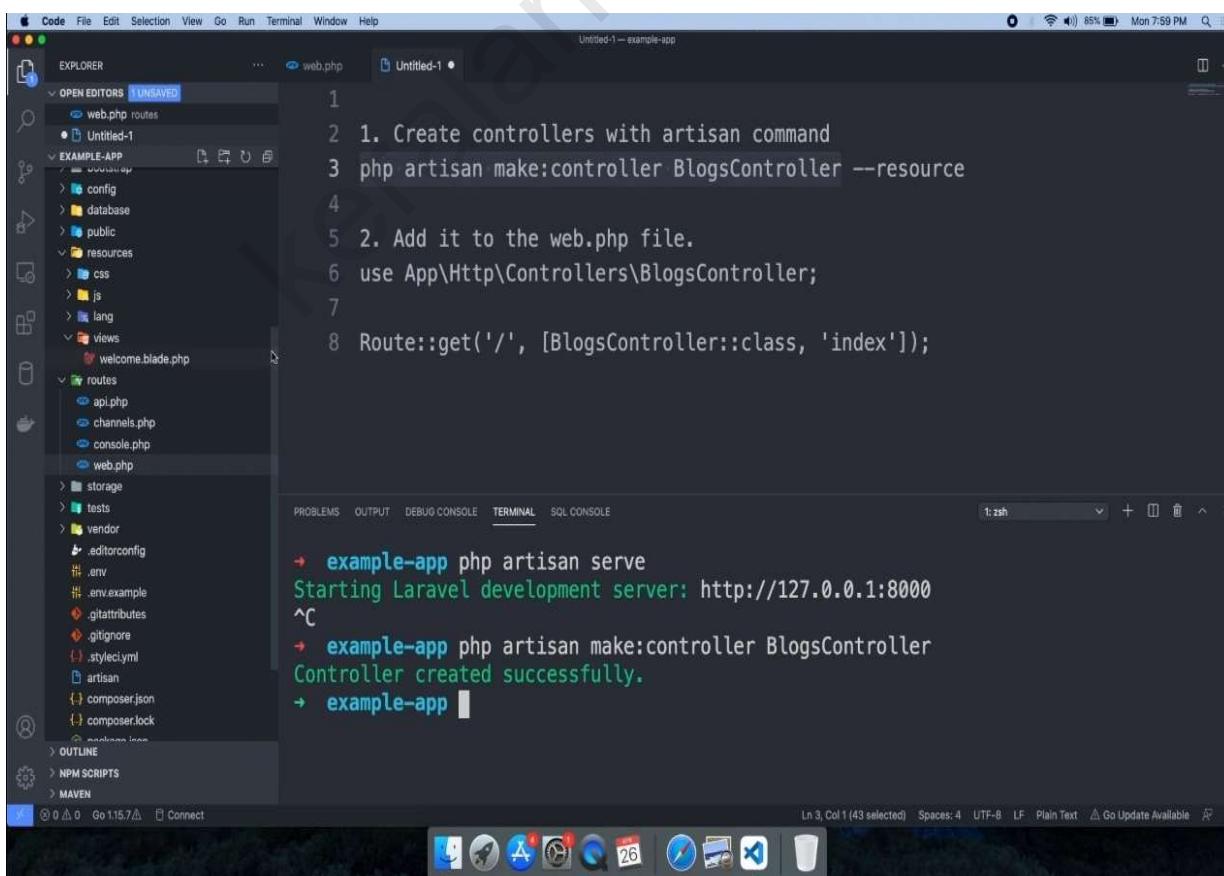




The screenshot shows the VS Code interface with the following details:

- Explorer:** Shows the project structure for an "EXAMPLE-APP". The "routes" folder contains "web.php".
- Editor:** The "web.php" file contains the following PHP code:


```
12 |     CONTAINS THE WED MIDDLEWARE GROUP. NOW CREATE SOMETHING GREAT!
13 |
14 */
15
16 Route::get('/', function () {
17     return view('welcome');
18 });
19 Route::get('/about', function () {
20     return "about page";
21 });
22
23
```
- Terminal:** Shows the command "php artisan serve" being run, and the response "Starting Laravel development server: http://127.0.0.1:8000".
- Bottom Bar:** Includes icons for file operations, a search bar, and a status bar indicating "Ln 20, Col 26 (21.m) Spaces: 4 UTF-8 LF Plain Text Go Update Available".

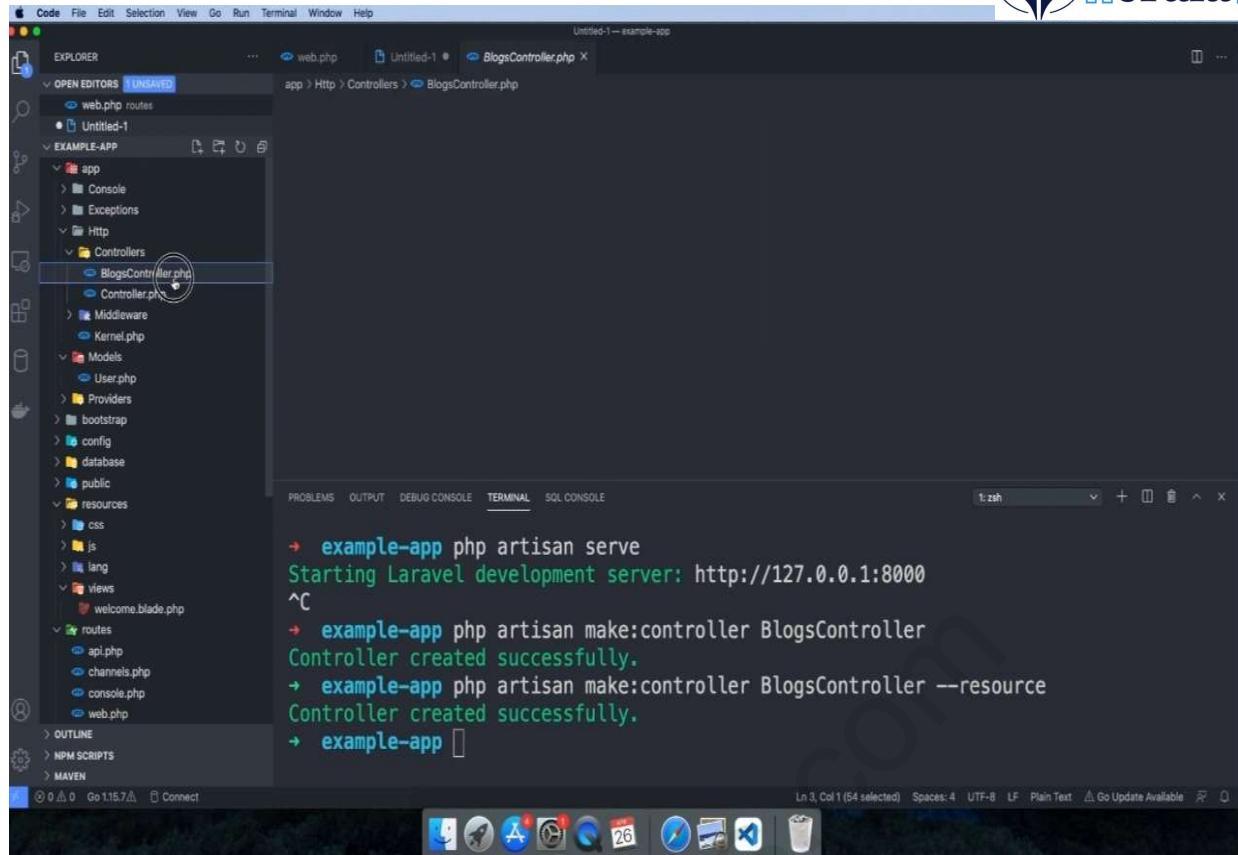


The screenshot shows the VS Code interface with the following details:

- Explorer:** Shows the project structure for an "EXAMPLE-APP". The "routes" folder contains "web.php".
- Editor:** The "web.php" file contains the following PHP code:


```
1
2 1. Create controllers with artisan command
3 php artisan make:controller BlogsController --resource
4
5 2. Add it to the web.php file.
6 use App\Http\Controllers\BlogsController;
7
8 Route::get('/', [BlogsController::class, 'index']);
```
- Terminal:** Shows the command "php artisan serve" being run, and the response "Starting Laravel development server: http://127.0.0.1:8000". It also shows the creation of a controller:


```
^C
→ example-app php artisan make:controller BlogsController
Controller created successfully.
→ example-app
```
- Bottom Bar:** Includes icons for file operations, a search bar, and a status bar indicating "Ln 3, Col 1 (43 selected) Spaces: 4 UTF-8 LF Plain Text Go Update Available".



The screenshot shows a Laravel application structure in the Explorer sidebar:

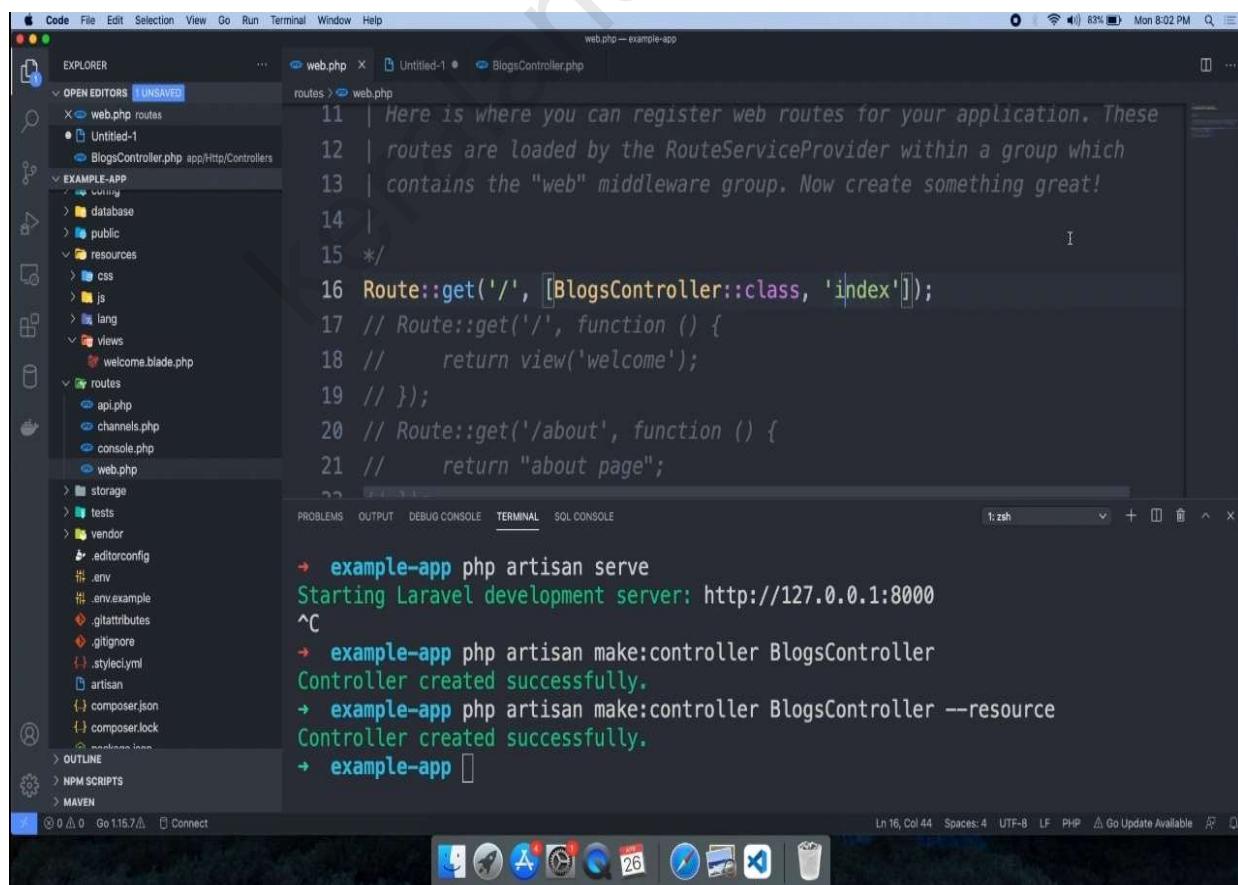
- OPEN EDITORS**: `web.php`, `Untitled-1`, `BlogsController.php`
- EXAMPLE-APP** folder:
 - app** folder:
 - Http** folder:
 - Controllers** folder: `BlogsController.php`, `Controller.php`
 - `Middleware`
 - `Kernel.php`
 - `Models`
 - `Providers`
 - `bootstrap`
 - `config`
 - `database`
 - `public`
 - `resources` folder:
 - `css`
 - `js`
 - `lang`
 - `views` folder: `welcome.blade.php`
 - `routes` folder:
 - `api.php`
 - `channels.php`
 - `console.php`
 - `web.php`
 - `OUTLINE`
 - `NPM SCRIPTS`
 - `MAVEN`

The **TERMINAL** tab shows the following command history:

```

→ example-app php artisan serve
Starting Laravel development server: http://127.0.0.1:8000
^C
→ example-app php artisan make:controller BlogsController
Controller created successfully.
→ example-app php artisan make:controller BlogsController --resource
Controller created successfully.
→ example-app

```



The screenshot shows the `routes/web.php` file in the code editor:

```

11 | Here is where you can register web routes for your application. These
12 | routes are loaded by the RouteServiceProvider within a group which
13 | contains the "web" middleware group. Now create something great!
14 |
15 */
16 Route::get('/', [BlogsController::class, 'index']);
17 // Route::get('/', function () {
18 //     return view('welcome');
19 // });
20 // Route::get('/about', function () {
21 //     return "about page";

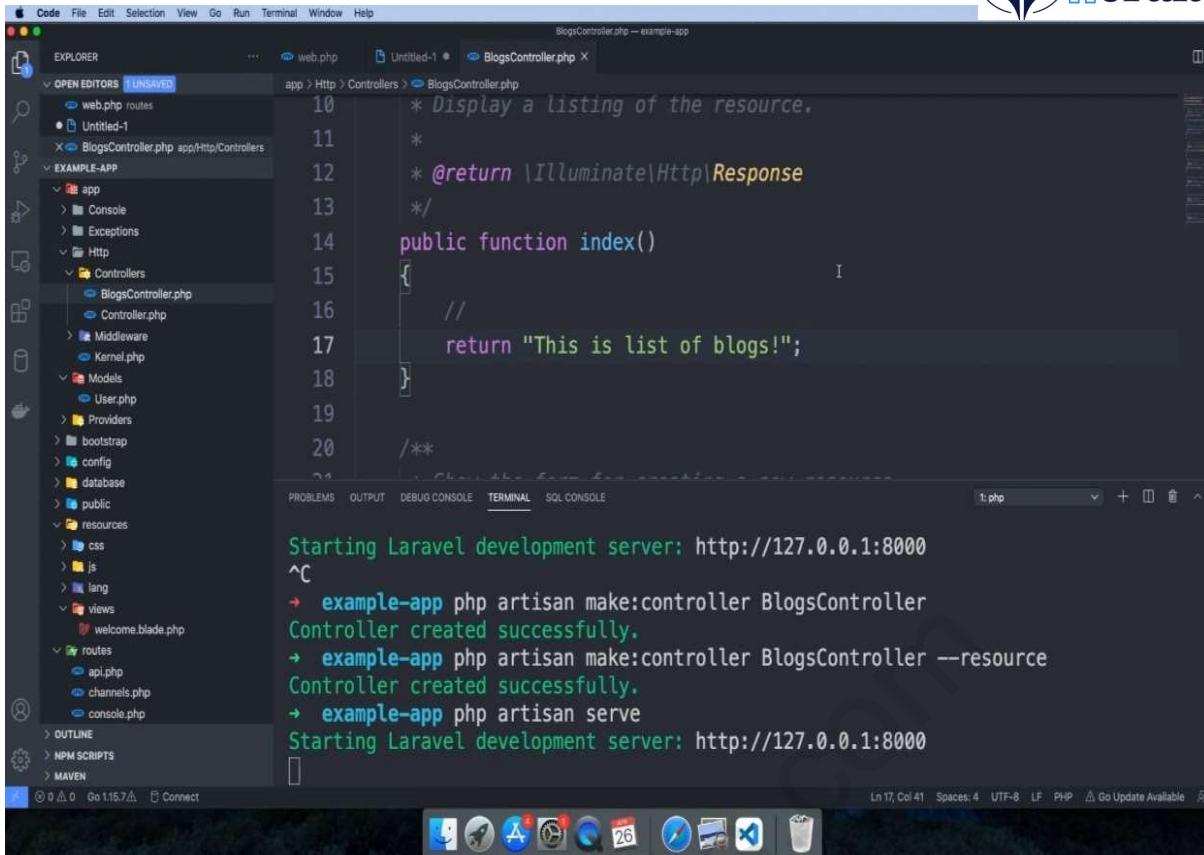
```

The **TERMINAL** tab shows the same command history as the previous screenshot:

```

→ example-app php artisan serve
Starting Laravel development server: http://127.0.0.1:8000
^C
→ example-app php artisan make:controller BlogsController
Controller created successfully.
→ example-app php artisan make:controller BlogsController --resource
Controller created successfully.
→ example-app

```

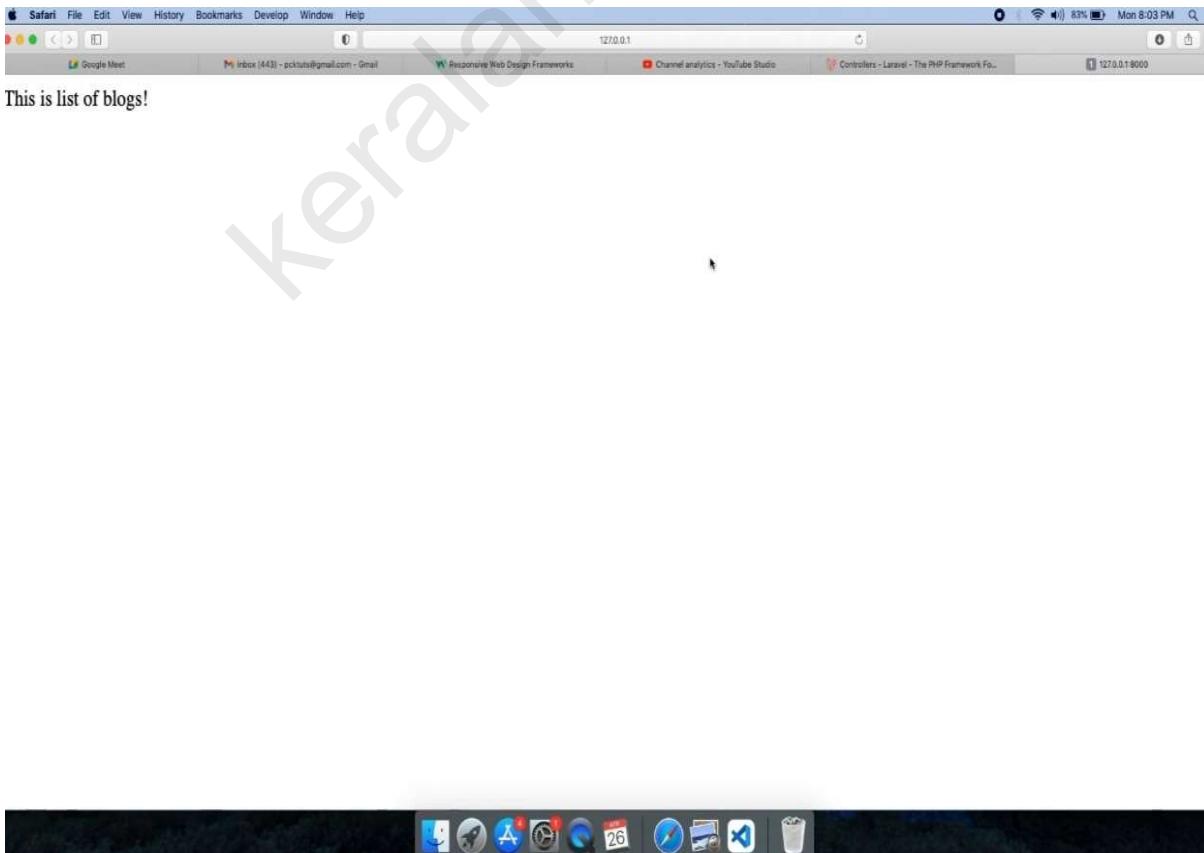


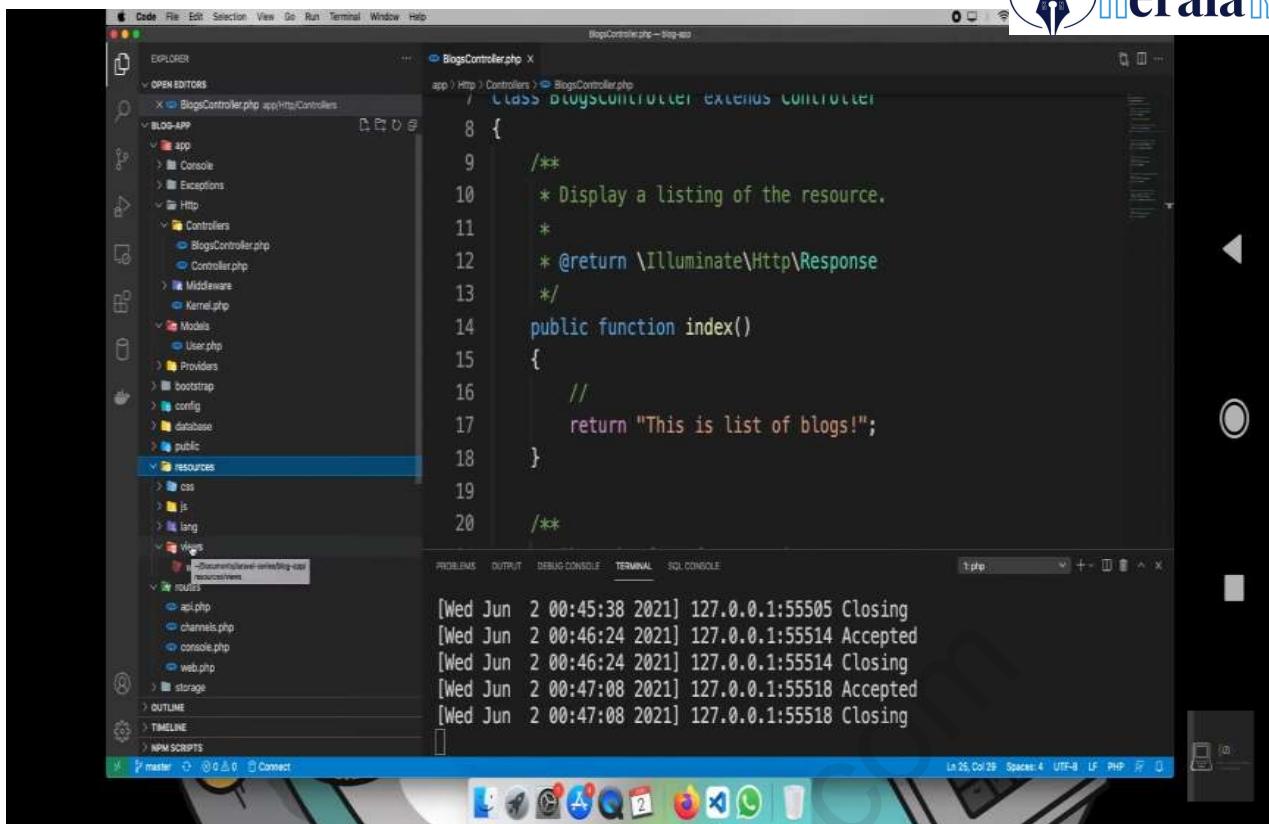
The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows the project structure under "EXAMPLE-APP".
- Code Editor:** Displays the content of `BlogsController.php` with syntax highlighting for PHP and annotations.
- Terminal:** Shows the command-line output of running artisan commands to create a controller and start the development server.
- Status Bar:** Shows the current file type as "1: php".

```
* Display a listing of the resource.
 *
 * @return \Illuminate\Http\Response
 */
public function index()
{
    //
    return "This is list of blogs!";
}

/**
Starting Laravel development server: http://127.0.0.1:8000
^C
→ example-app php artisan make:controller BlogsController
Controller created successfully.
→ example-app php artisan make:controller BlogsController --resource
Controller created successfully.
→ example-app php artisan serve
Starting Laravel development server: http://127.0.0.1:8000
```



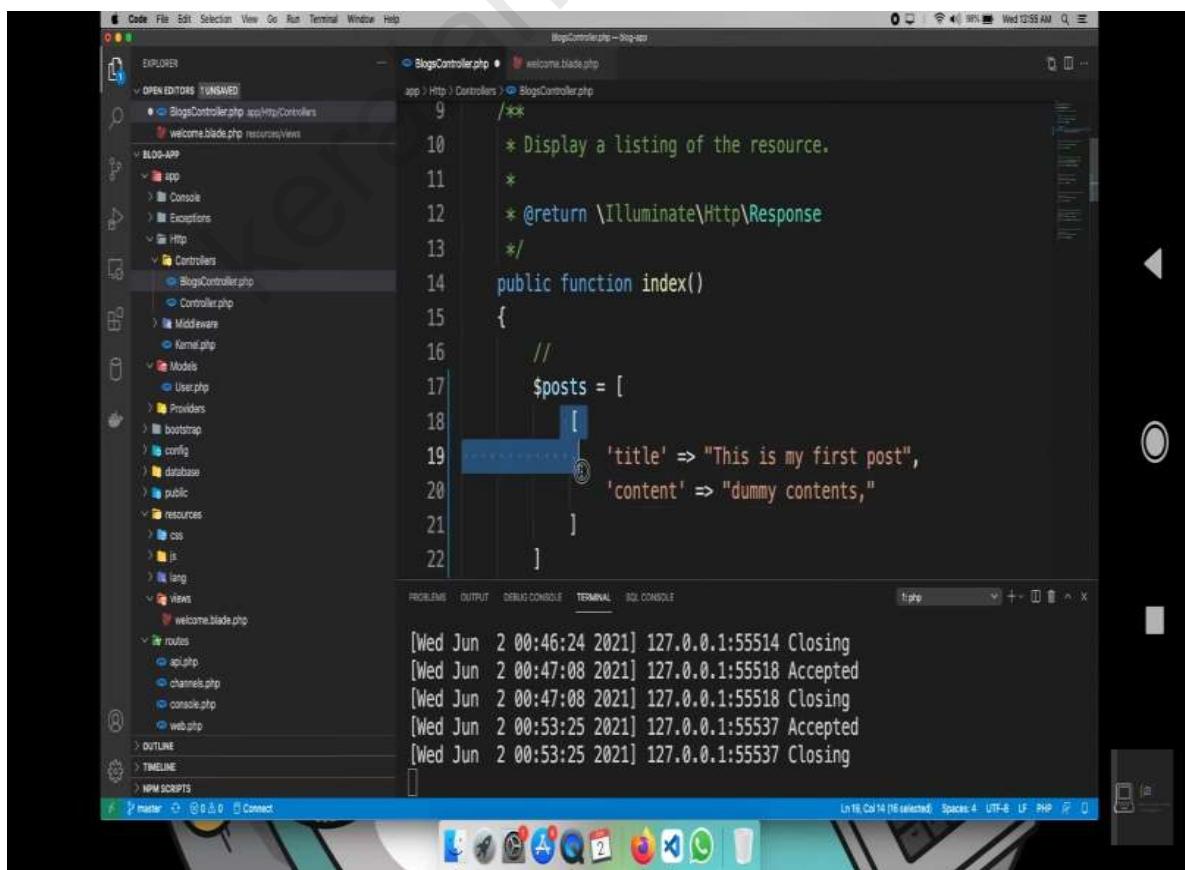


```

class BlogsController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        //
        return "This is list of blogs!";
    }
}

```

[Wed Jun 2 00:45:38 2021] 127.0.0.1:55505 Closing
[Wed Jun 2 00:46:24 2021] 127.0.0.1:55514 Accepted
[Wed Jun 2 00:46:24 2021] 127.0.0.1:55514 Closing
[Wed Jun 2 00:47:08 2021] 127.0.0.1:55518 Accepted
[Wed Jun 2 00:47:08 2021] 127.0.0.1:55518 Closing

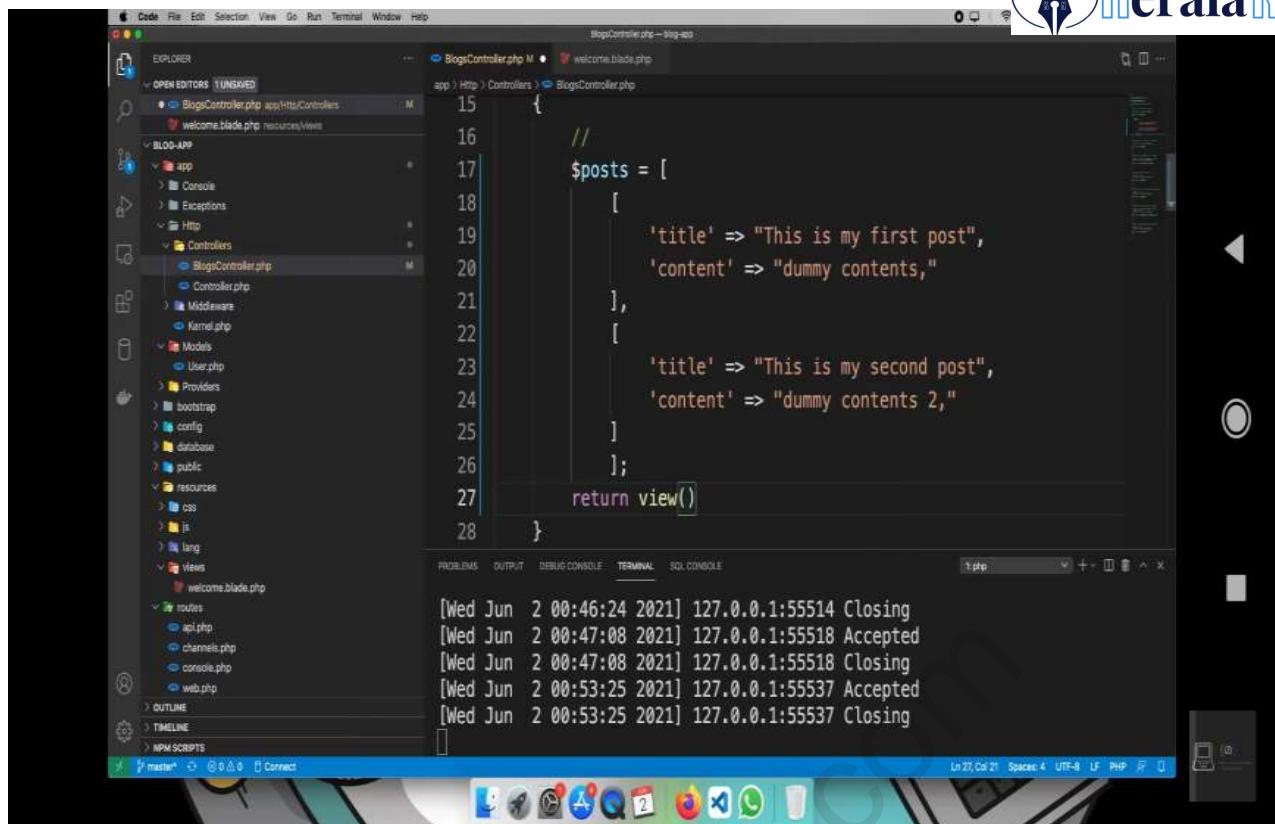


```

class BlogsController extends Controller
{
    /**
     * Display a listing of the resource.
     *
     * @return \Illuminate\Http\Response
     */
    public function index()
    {
        //
        $posts = [
            [
                'title' => "This is my first post",
                'content' => "dummy contents,"
            ]
        ];
        return response()->json($posts);
    }
}

```

[Wed Jun 2 00:46:24 2021] 127.0.0.1:55514 Closing
[Wed Jun 2 00:47:08 2021] 127.0.0.1:55518 Accepted
[Wed Jun 2 00:47:08 2021] 127.0.0.1:55518 Closing
[Wed Jun 2 00:53:25 2021] 127.0.0.1:55537 Accepted
[Wed Jun 2 00:53:25 2021] 127.0.0.1:55537 Closing

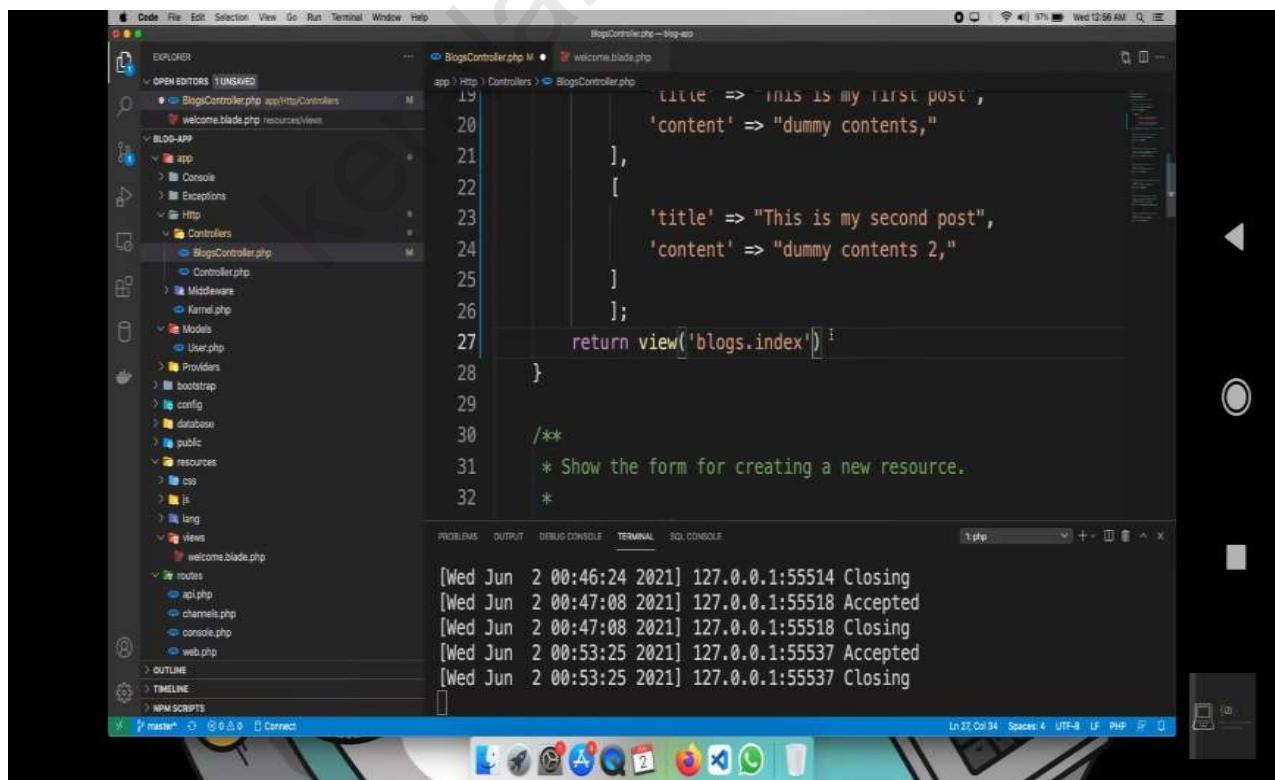


```

15  {
16      // 
17      $posts = [
18          [
19              'title' => "This is my first post",
20              'content' => "dummy contents,"
21          ],
22          [
23              'title' => "This is my second post",
24              'content' => "dummy contents 2,"
25          ]
26      ];
27      return view()
28  }

```

[Wed Jun 2 00:46:24 2021] 127.0.0.1:55514 Closing
 [Wed Jun 2 00:47:08 2021] 127.0.0.1:55518 Accepted
 [Wed Jun 2 00:47:08 2021] 127.0.0.1:55518 Closing
 [Wed Jun 2 00:53:25 2021] 127.0.0.1:55537 Accepted
 [Wed Jun 2 00:53:25 2021] 127.0.0.1:55537 Closing

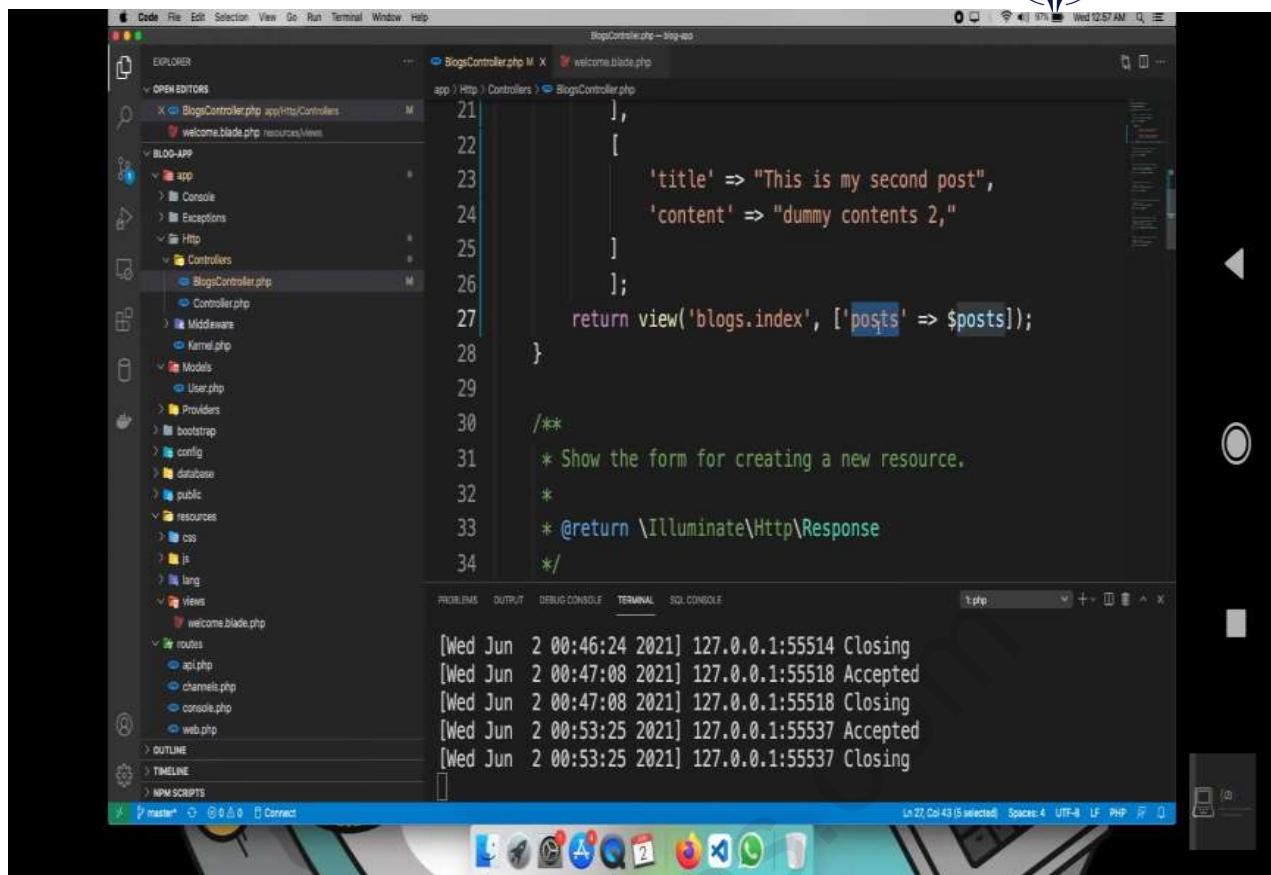


```

19  {
20      'title' => "THIS IS MY FIRST POST",
21      'content' => "dummy contents,"
22  ],
23  [
24      'title' => "This is my second post",
25      'content' => "dummy contents 2,"
26  ];
27  return view('blogs.index')
28 }

```

[Wed Jun 2 00:46:24 2021] 127.0.0.1:55514 Closing
 [Wed Jun 2 00:47:08 2021] 127.0.0.1:55518 Accepted
 [Wed Jun 2 00:47:08 2021] 127.0.0.1:55518 Closing
 [Wed Jun 2 00:53:25 2021] 127.0.0.1:55537 Accepted
 [Wed Jun 2 00:53:25 2021] 127.0.0.1:55537 Closing



The screenshot shows a Microsoft Visual Studio Code (VS Code) interface with the following details:

- File Structure:** The left sidebar displays the project structure of a Laravel application named "BLOG-APP". It includes the "app", "Http", "Controllers", "Middleware", "Kernel.php", "Models", "User.php", "Providers", "bootstrap", "config", "database", "public", "resources" (with "css", "js", "lang", "views" subfolders), and "routes" (with "api.php", "channels.php", "console.php", and "web.php").
- Code Editor:** The main editor window contains the "BlogsController.php" file. The code is as follows:

```
    21     ],
    22     [
    23         'title' => "This is my second post",
    24         'content' => "dummy contents 2,"
    25     ]
    26 ];
    27     return view('blogs.index', ['posts' => $posts]);
    28 }
    29 /**
    30 * Show the form for creating a new resource.
    31 *
    32 * @return \Illuminate\Http\Response
    33 */
    34 */
```

- Terminal:** The bottom right terminal window shows the following log output:

```
[Wed Jun 2 00:46:24 2021] 127.0.0.1:55514 Closing
[Wed Jun 2 00:47:08 2021] 127.0.0.1:55518 Accepted
[Wed Jun 2 00:47:08 2021] 127.0.0.1:55518 Closing
[Wed Jun 2 00:53:25 2021] 127.0.0.1:55537 Accepted
[Wed Jun 2 00:53:25 2021] 127.0.0.1:55537 Closing
```