# 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).


- 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 by default. 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

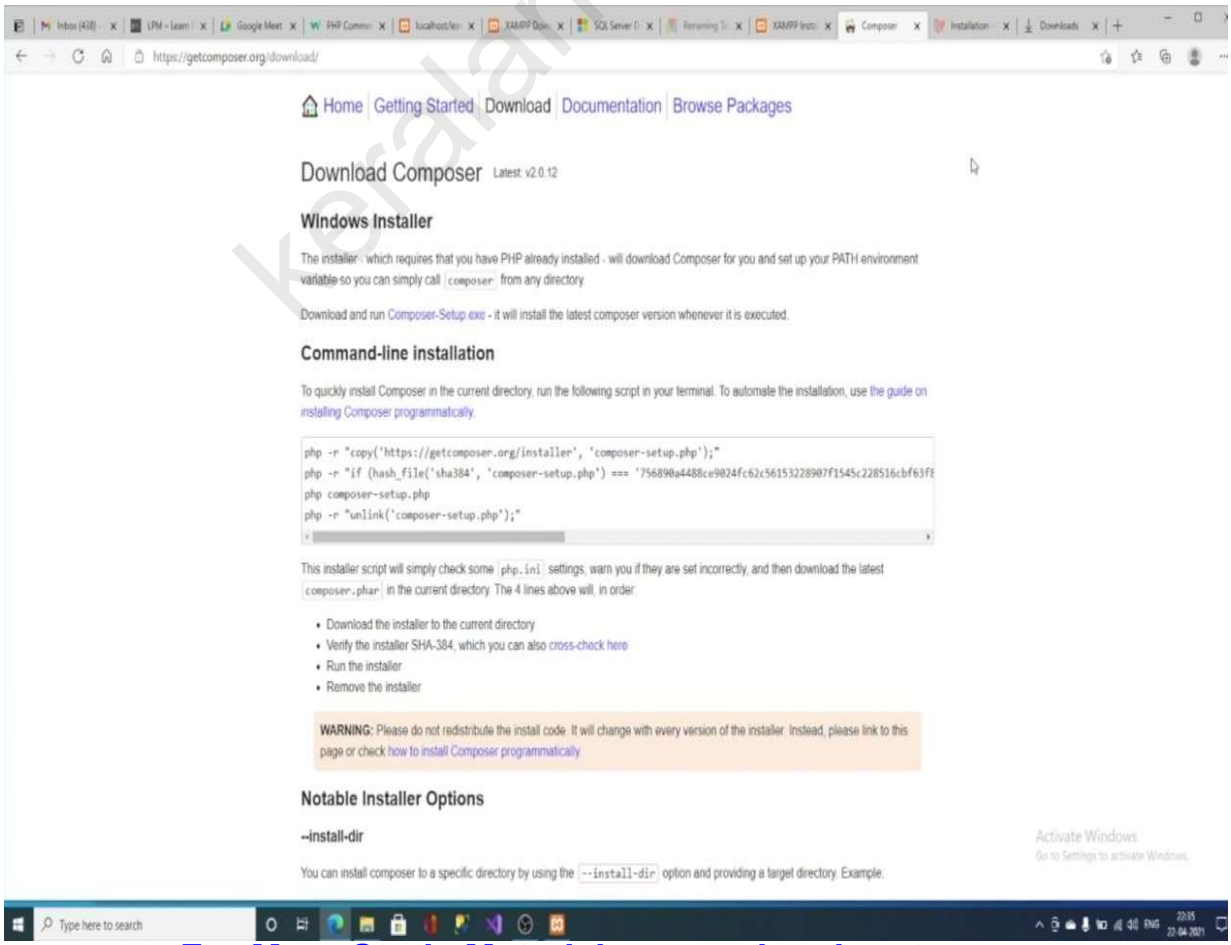
# PHP - Looping Through the Values

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

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

$obj = json_decode($jsonobj);

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

</body>
</html>
```

Output

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

# Laravel installation

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

string variable to choose which services should be configured in your new application's `docker-compose.yml` file. Available services include `mysql`, `pgsql`, `mariadb`, `redis`, `memcached`, `meilisearch`, `selenium`, and `mailhog`:

```
curl -s "https://laravel.build/example-app?with=mysql,redis" | bash
```

If you do not specify which services you would like configured, a default stack of `mysql`, `redis`, `meilisearch`, `mailhog`, 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
```



Type here to search

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

string variable to choose which services should be configured in your new application's `docker-compose.yml` file. Available services include `mysql`, `pgsql`, `mariadb`, `redis`, `memcached`, `meilisearch`, `selenium`, and `mailhog`:

```
curl -s "https://laravel.build/example-app?with=mysql,redis" | bash
```

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

## # Installation Via Composer

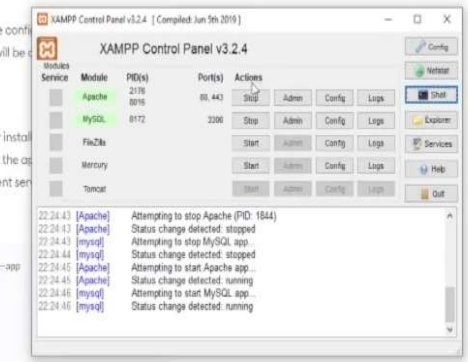
If your computer already has PHP and Composer installed, you may create a new Laravel project by using Composer directly. After the application has been created, you may start Laravel's local development server using the Artisan CLI's `serve` command:

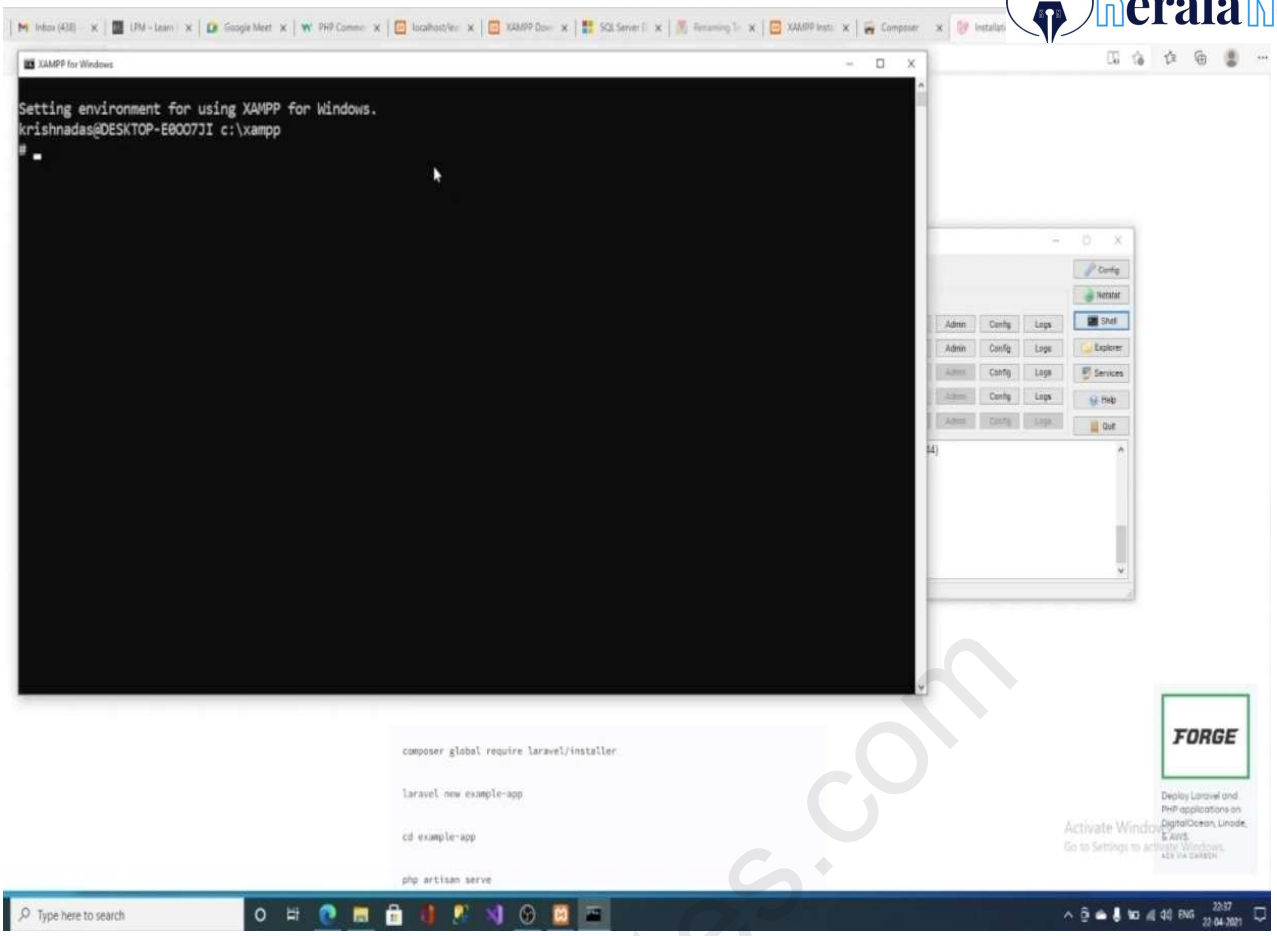
```
composer create-project laravel/laravel example-app

cd example-app

php artisan serve
```

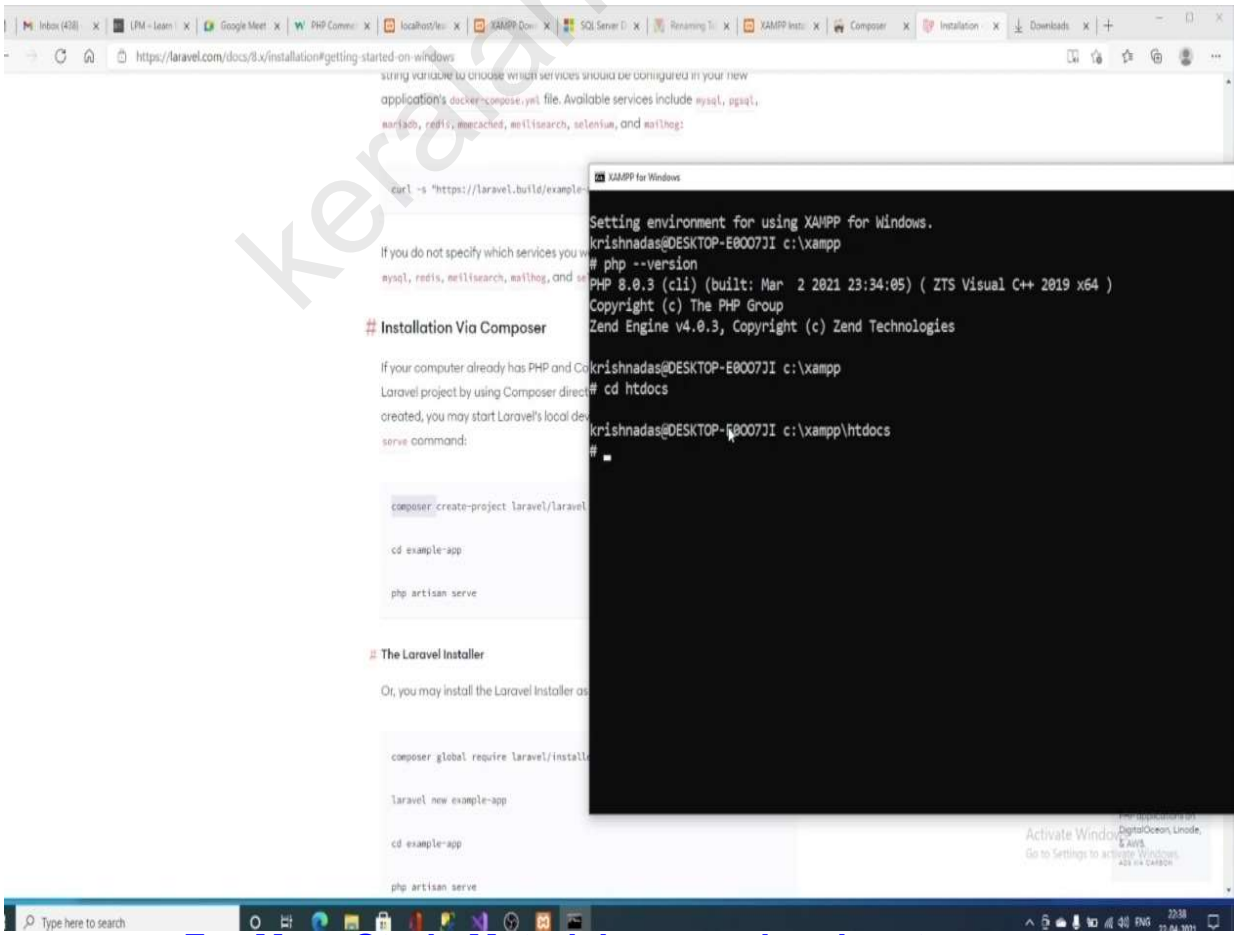
### # The Laravel Installer

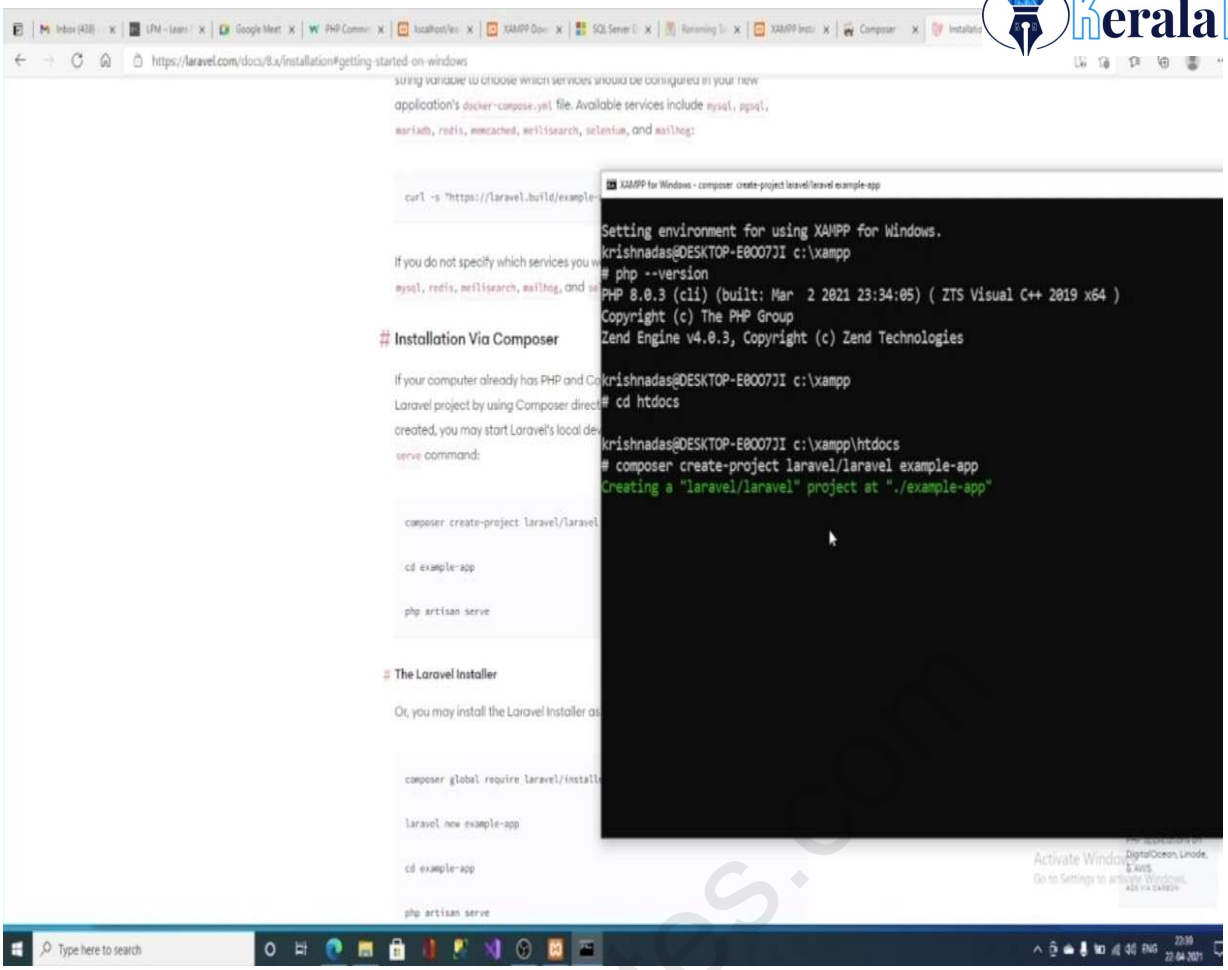
Or, you may install the Laravel Installer as a global Composer dependency:

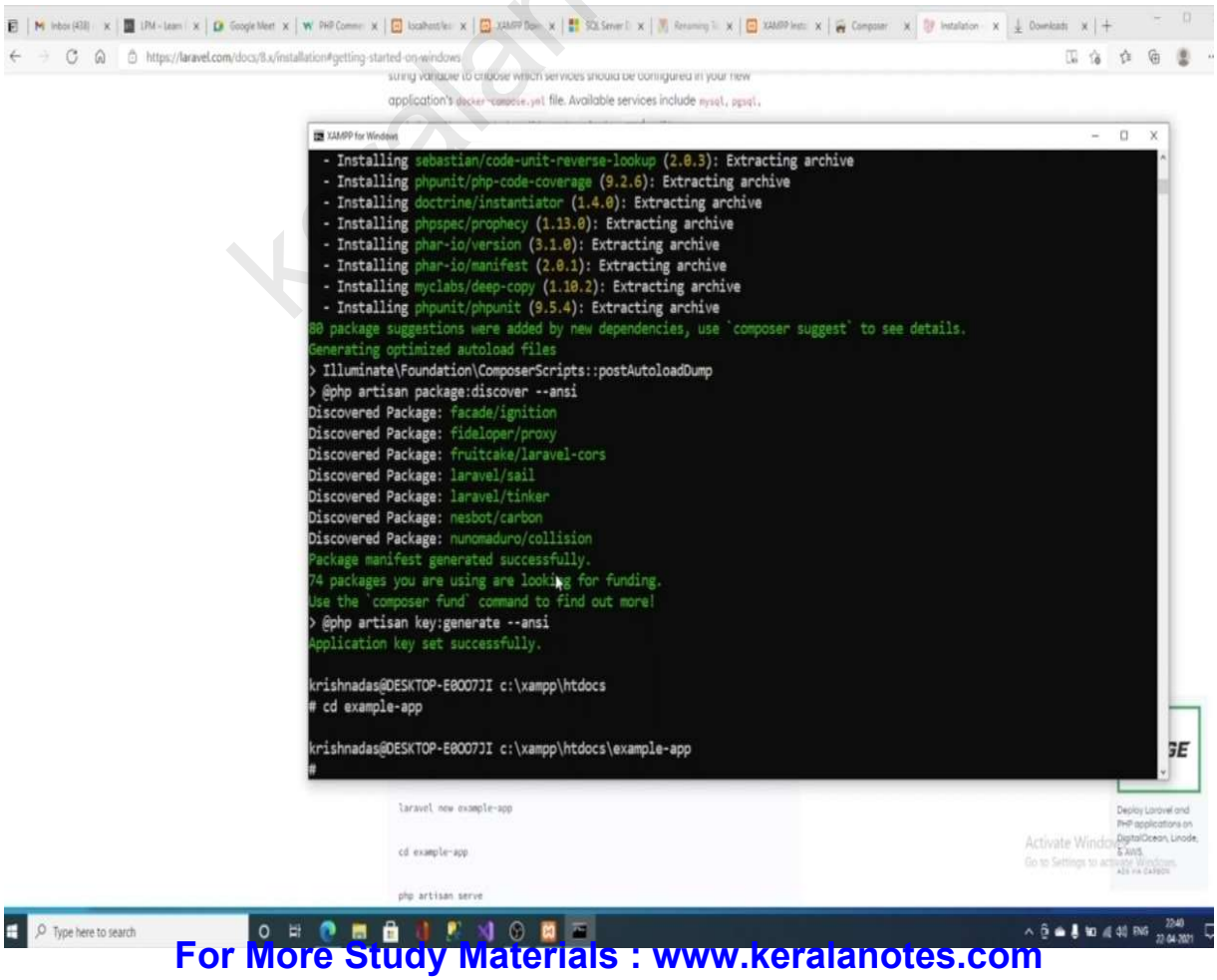
```
composer global require laravel/installer

laravel new example-app

cd example-app

php artisan serve
```







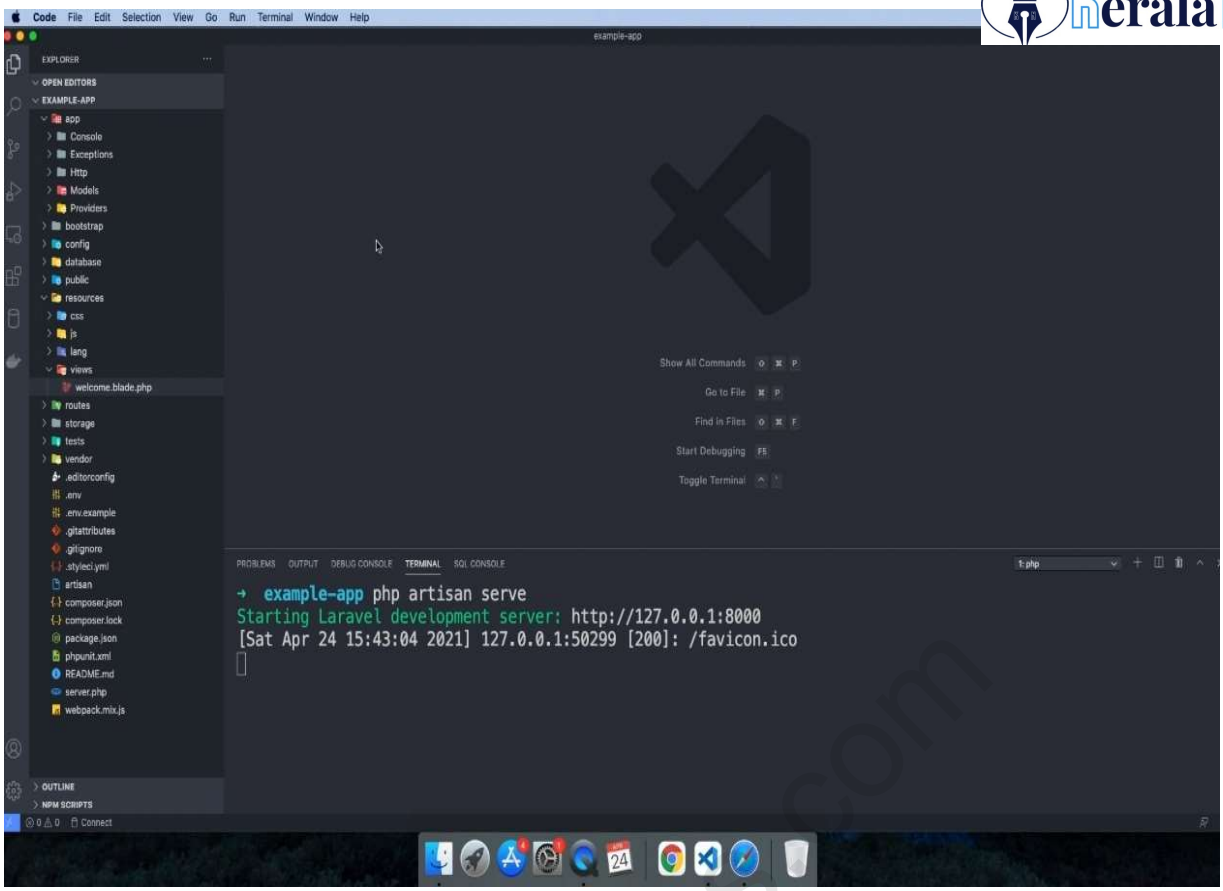
# Features of Laravel

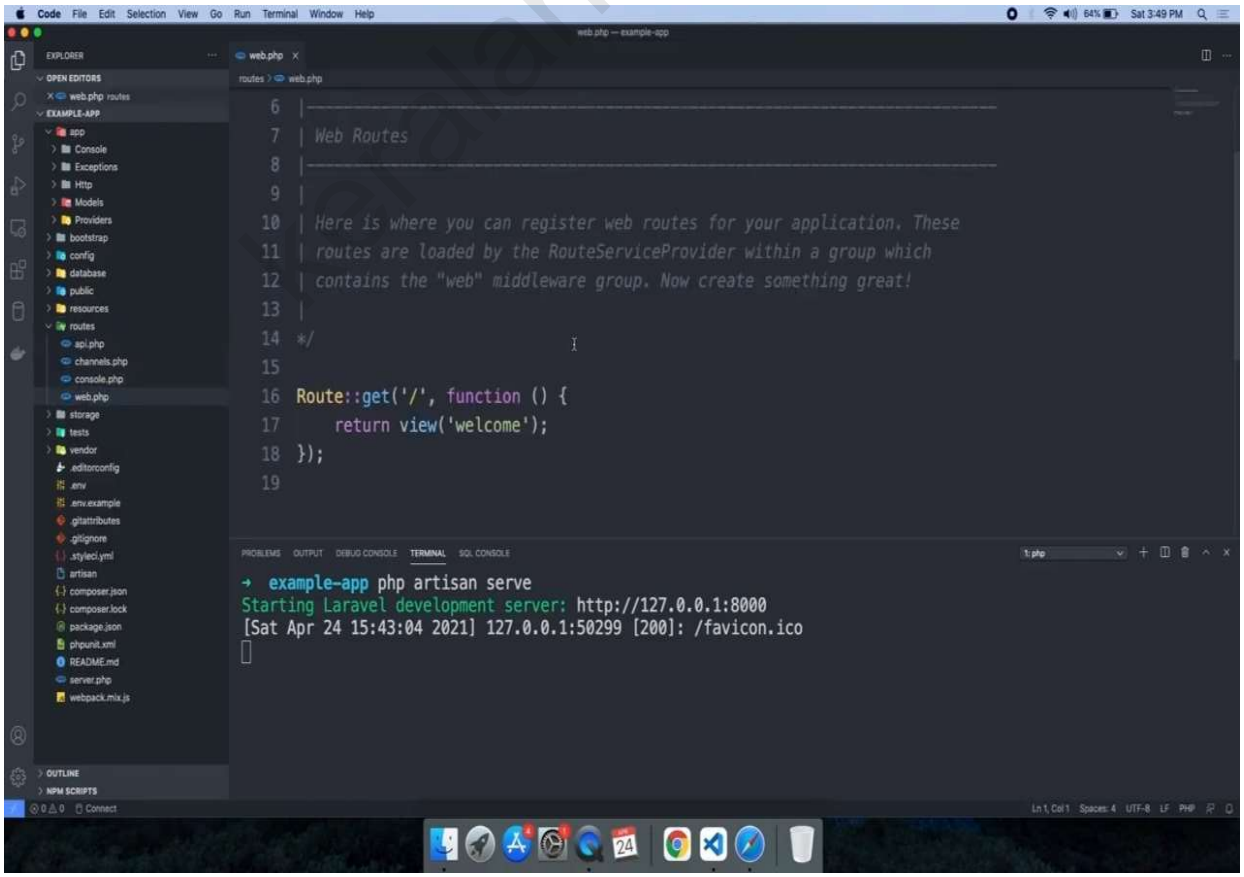
- Template Engine (blade)
- MVC Architecture
- Eloquent ORM (Object Relational Mapping)
- Security
- Artisan
- Libraries & Modular
- Database Migration System
- Unit-Testing

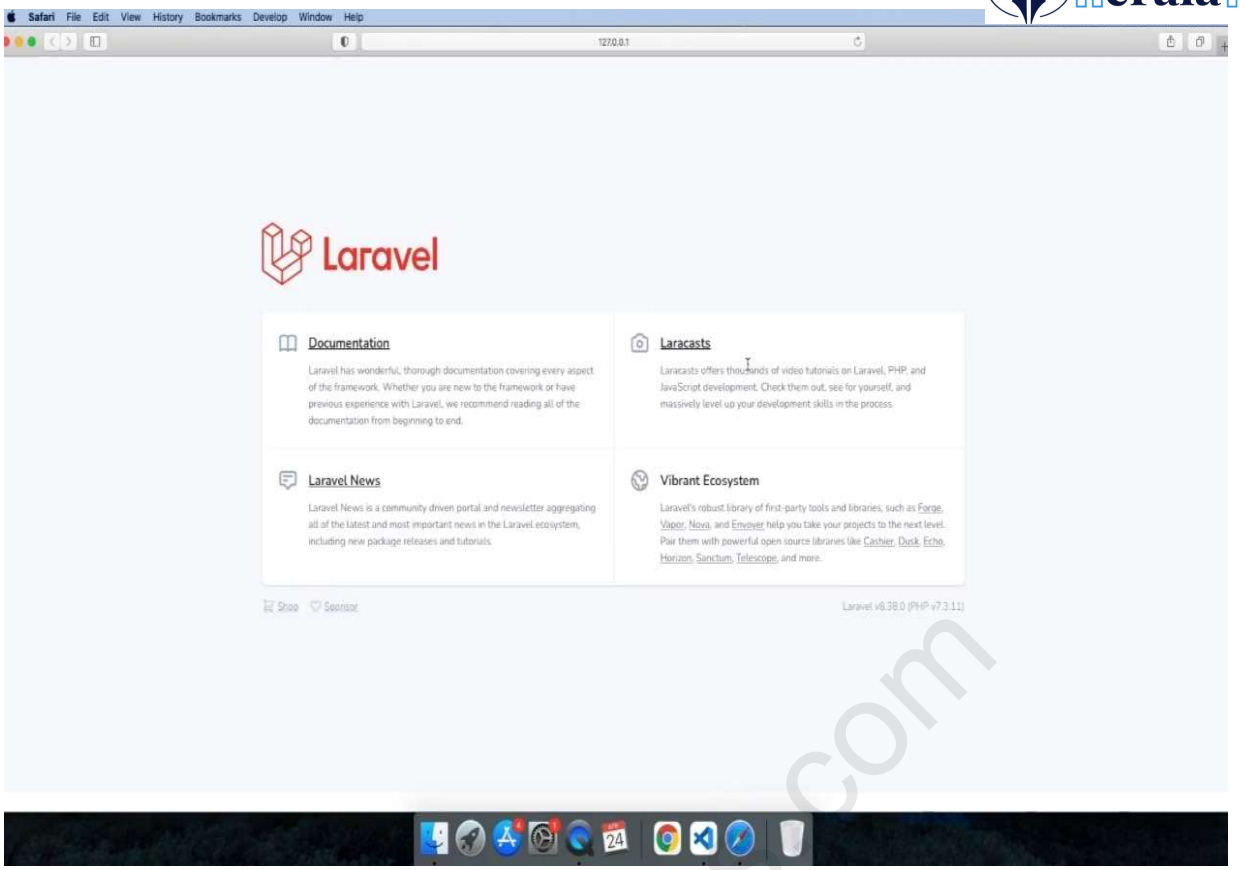


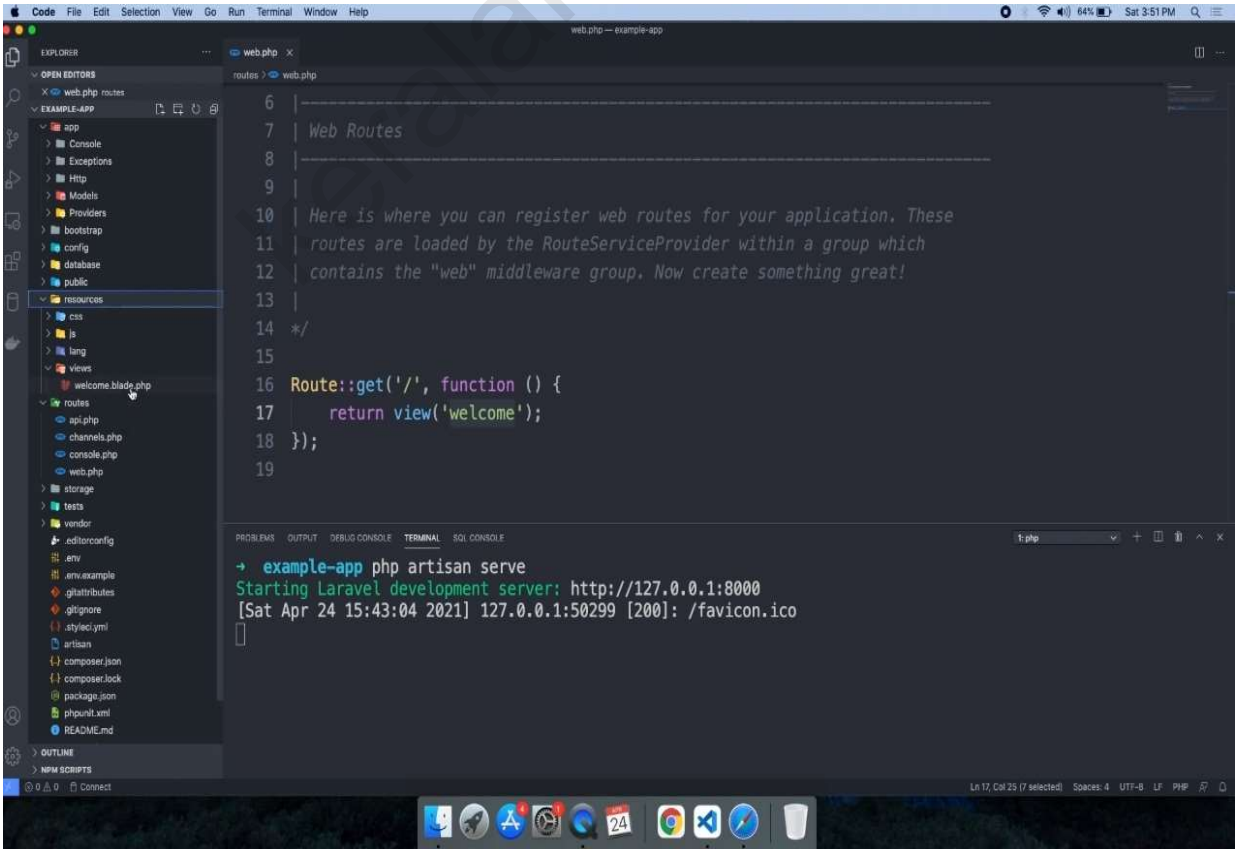
# Prerequisite for learning laravel

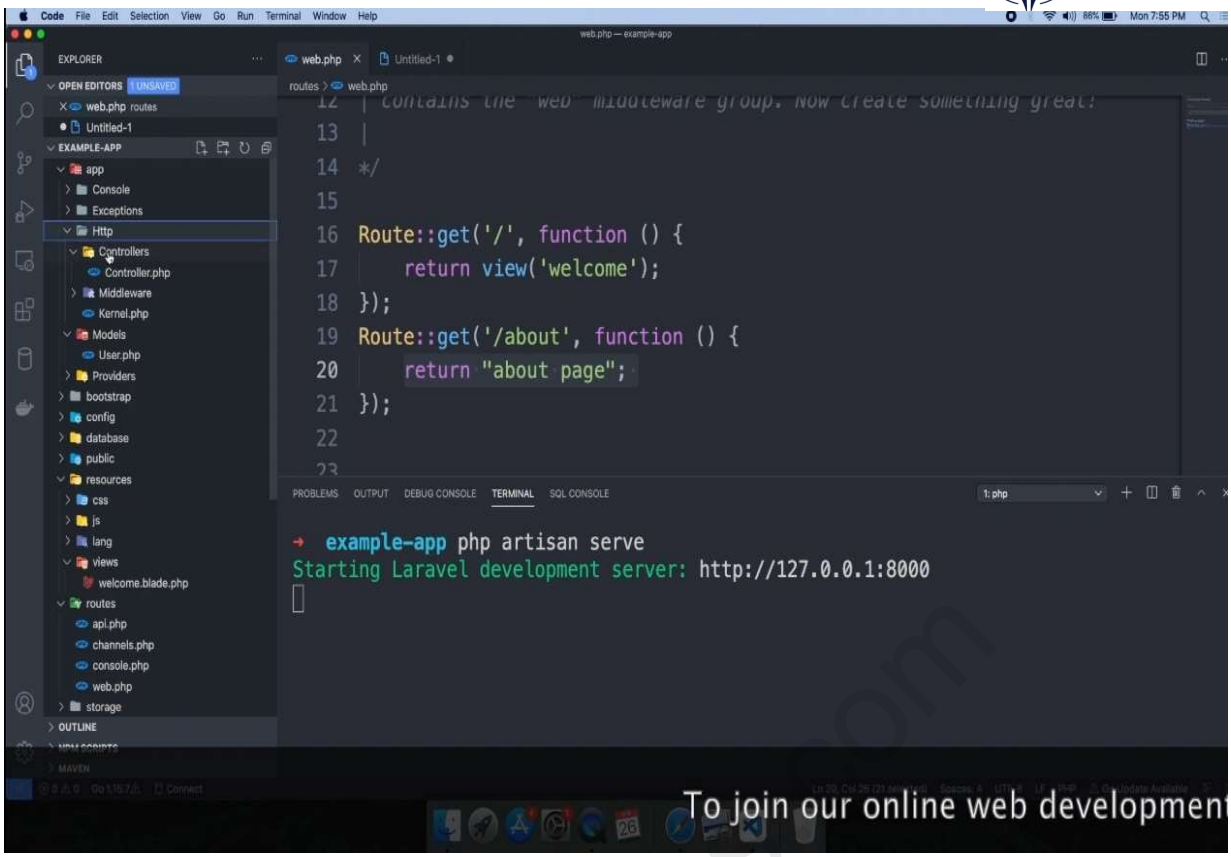
- Good knowledge of PHP
- Intermediate knowledge of SQL such as MySQL
- Basic knowledge of HTML, CSS & Javascript
- Familiar with command line/linux is also helpful
- For working in real projects learning Git is also mandatory

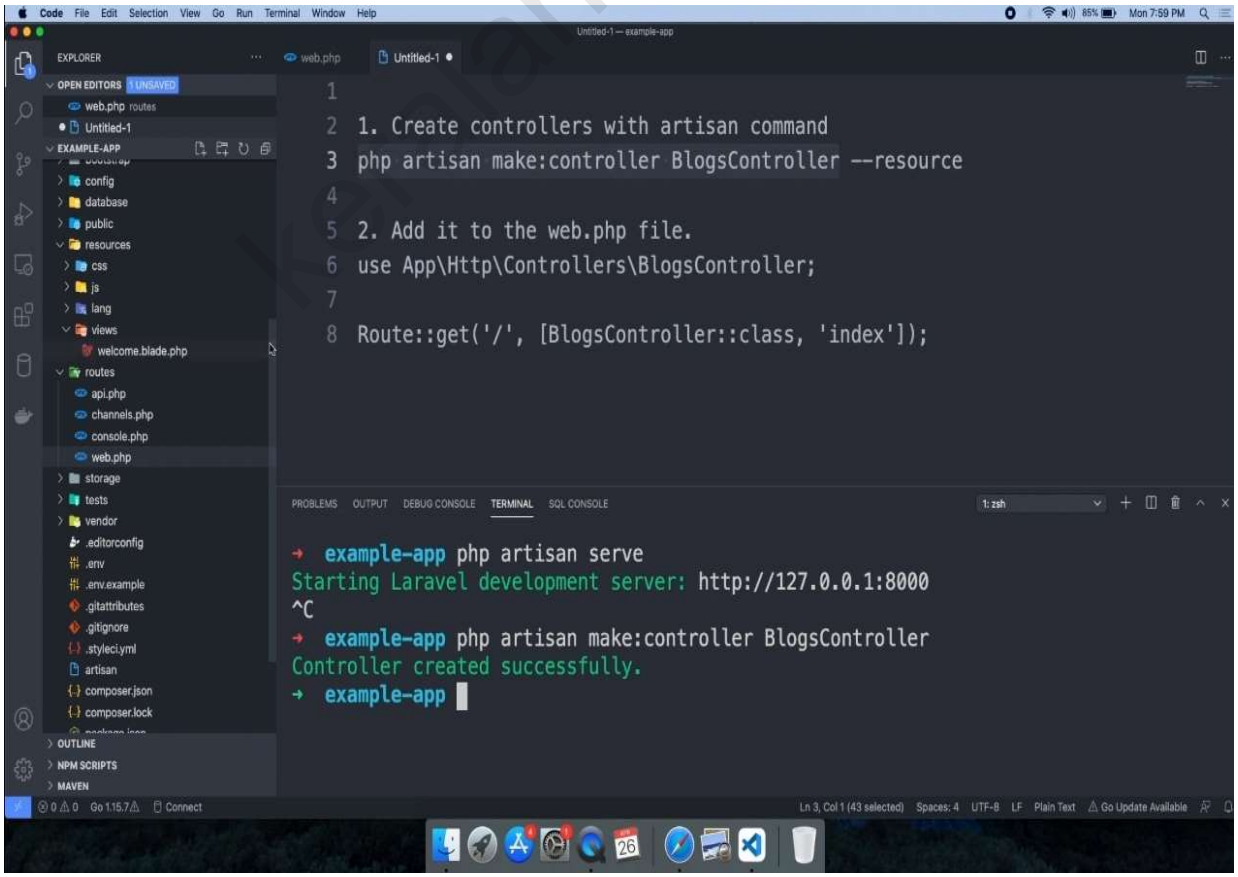


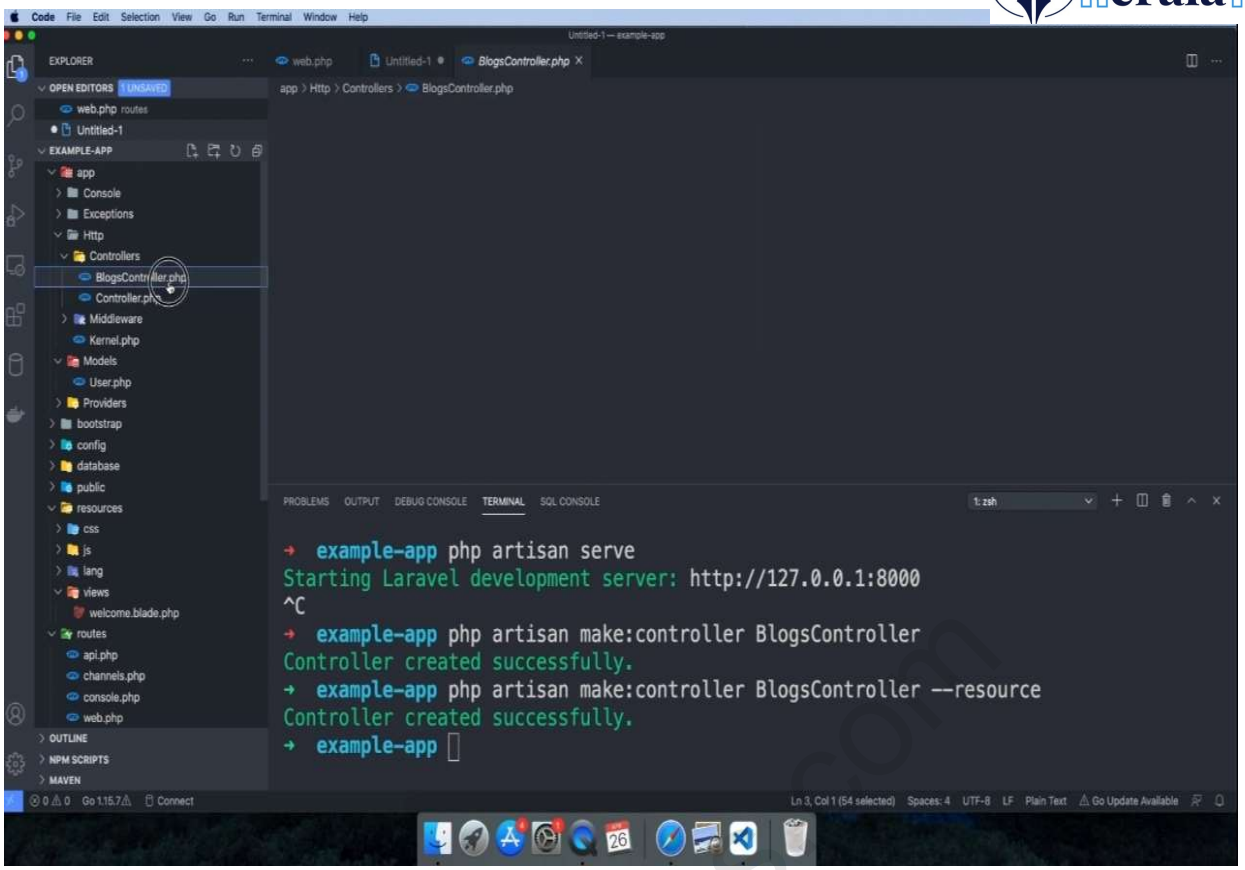



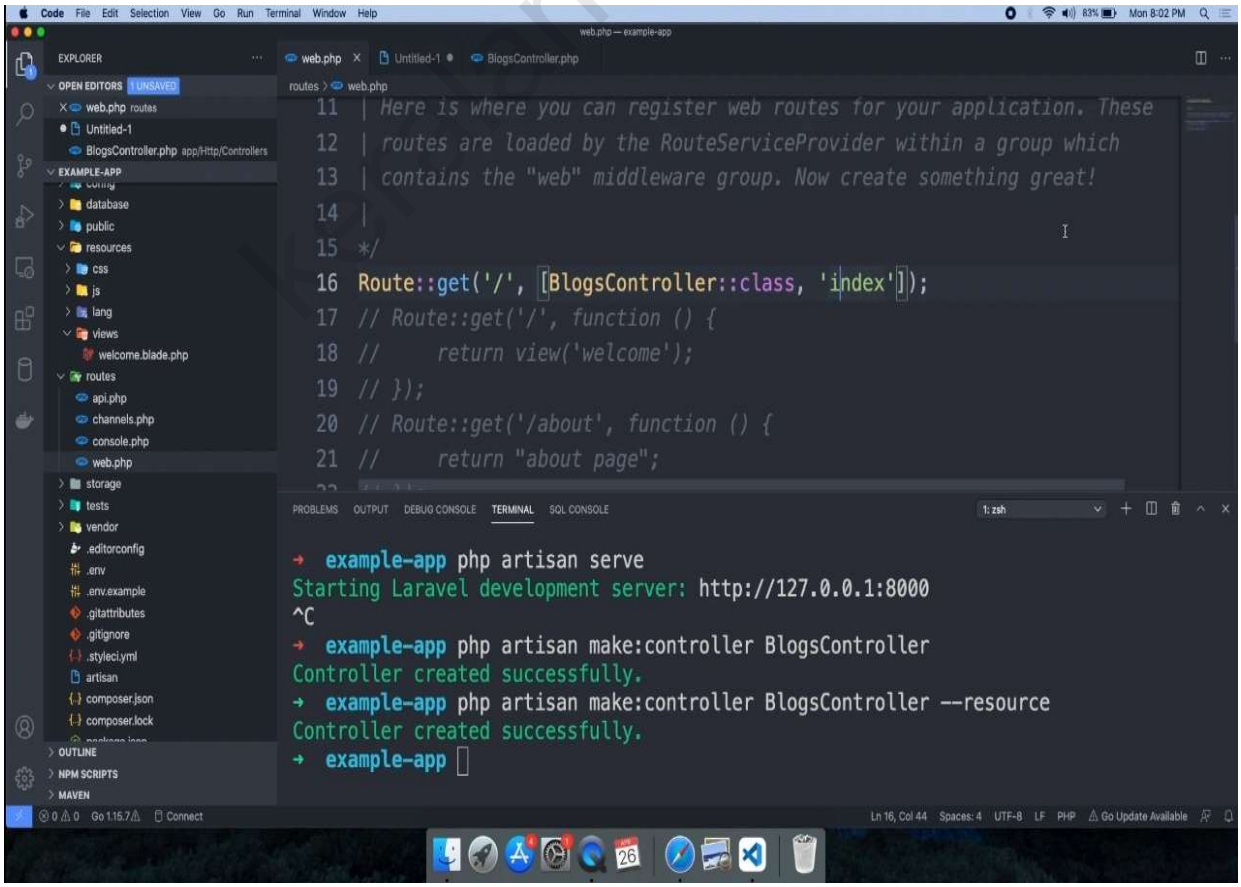


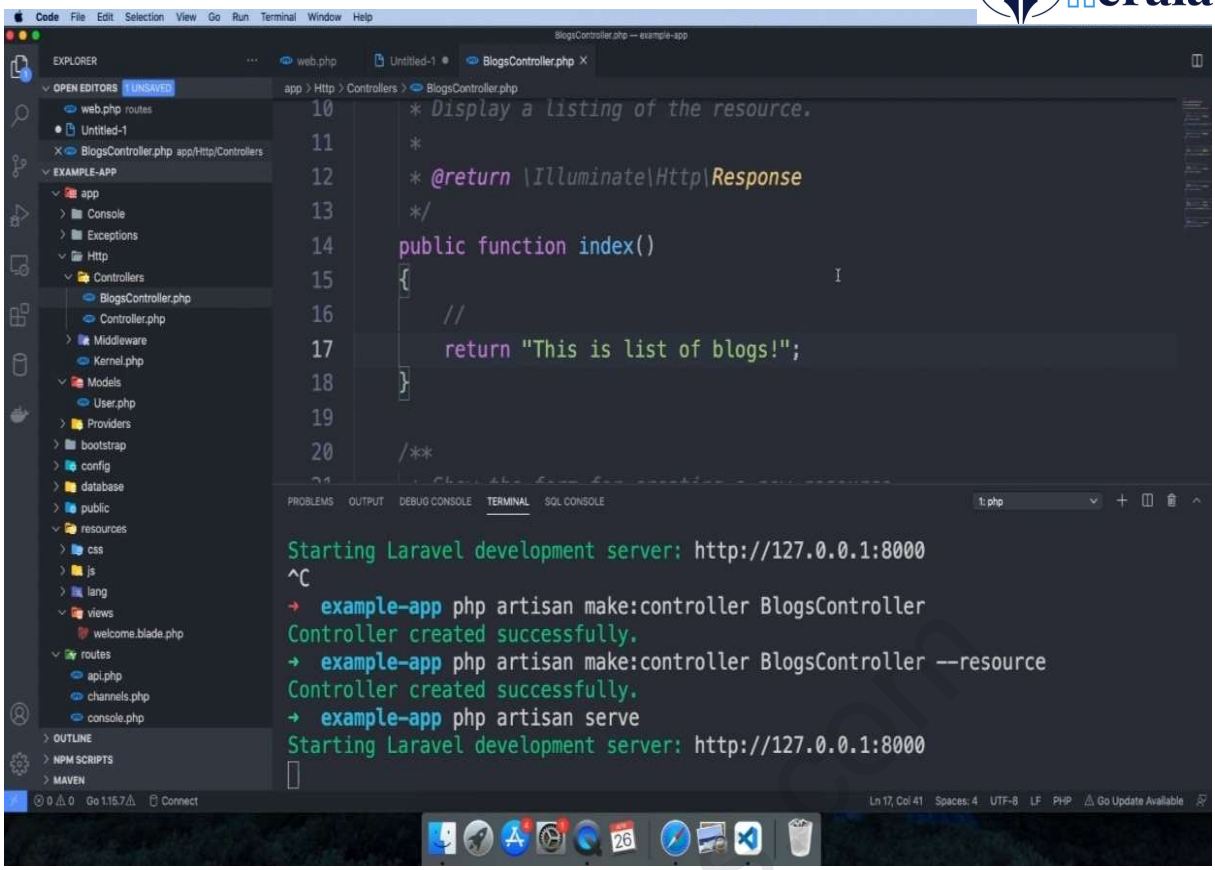




web.php
Untitled-1
routes > web.php
| contains the "web" middleware group. Now create something great!
*/
Route::get('/', function () {
return view('welcome');
});
Route::get('/about', function () {
return "about page";
});
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE
1: php
example-app php artisan serve
Starting Laravel development server: http://127.0.0.1:8000
To join our online web development


1. Create controllers with artisan command
php artisan make:controller BlogsController --resource
2. Add it to the web.php file.
use App\Http\Controllers\BlogsController;
Route::get('/', [BlogsController::class, 'index']);
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE
1: zsh
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

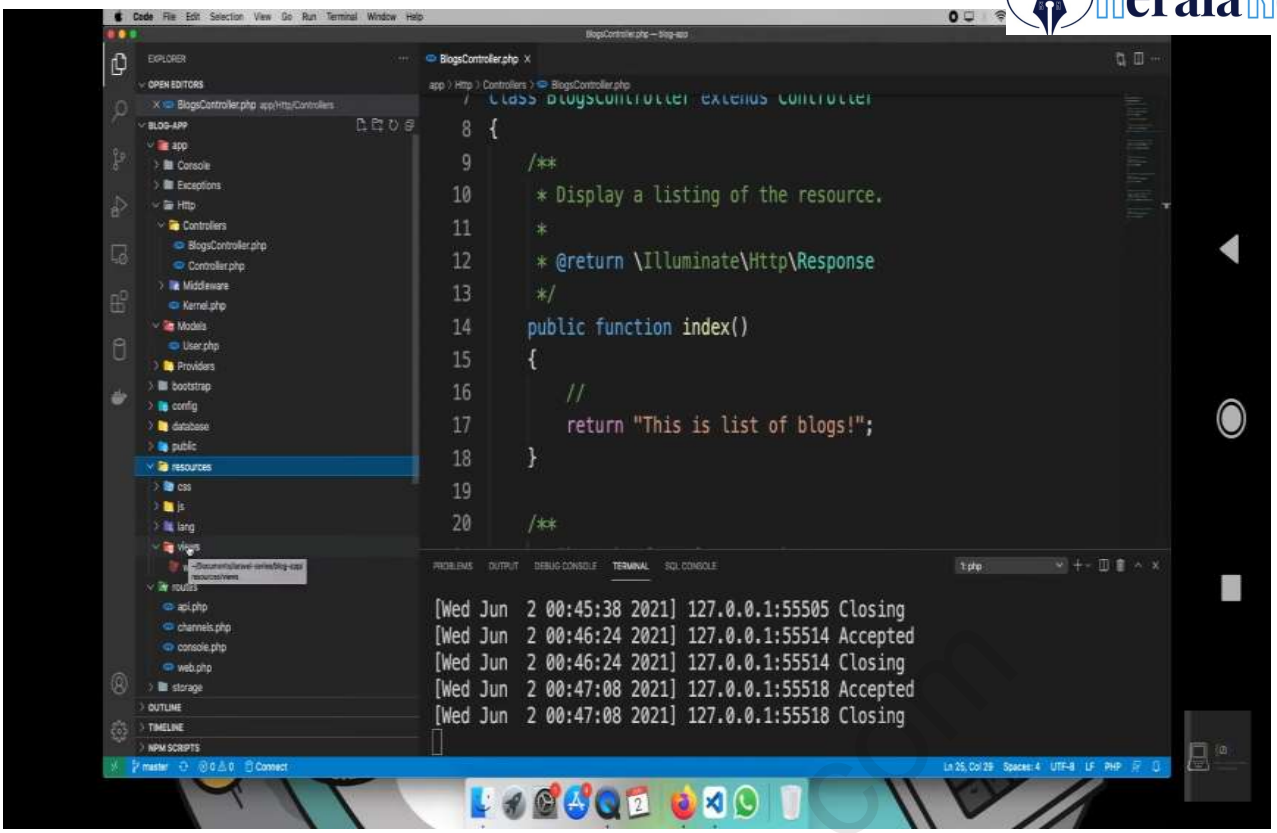

EXPLORER
OPEN EDITORS 1 UNSAVED
web.php routes
Untitled-1
EXAMPLE-APP
app
Console
Exceptions
Http
Controllers
BlogsController.php
Controller.php
Middleware
Kernel.php
Models
User.php
Providers
bootstrap
config
database
public
resources
css
js
lang
views
welcome.blade.php
routes
api.php
channels.php
console.php
web.php
OUTLINE
NPM SCRIPTS
MAVEN
app > Http > Controllers > BlogsController.php
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE
1: zsh
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

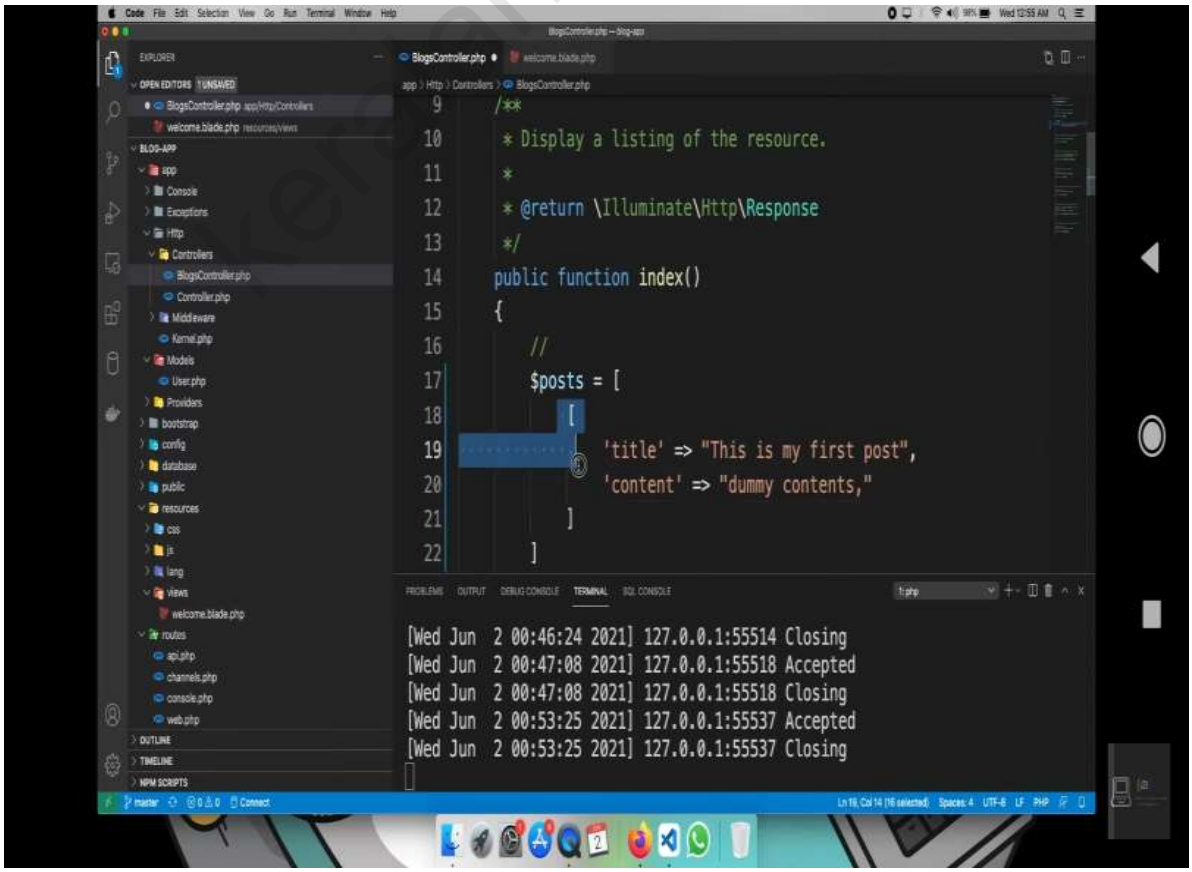

EXPLORER
OPEN EDITORS 1 UNSAVED
web.php routes
Untitled-1
BlogsController.php app/Http/Controllers
EXAMPLE-APP
database
public
resources
css
js
lang
views
welcome.blade.php
routes
api.php
channels.php
console.php
web.php
storage
tests
vendor
.editorconfig
.env
.env.example
.gitattributes
.gitignore
.styleci.yml
artisan
composer.json
composer.lock
OUTLINE
NPM SCRIPTS
MAVEN
routes > web.php
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider within a group which
| contains the "web" middleware group. Now create something great!
|
*/
Route::get('/', [BlogsController::class, 'index']);
// Route::get('/', function () {
//     return view('welcome');
// });
// Route::get('/about', function () {
//     return "about page";
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE
1: zsh
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

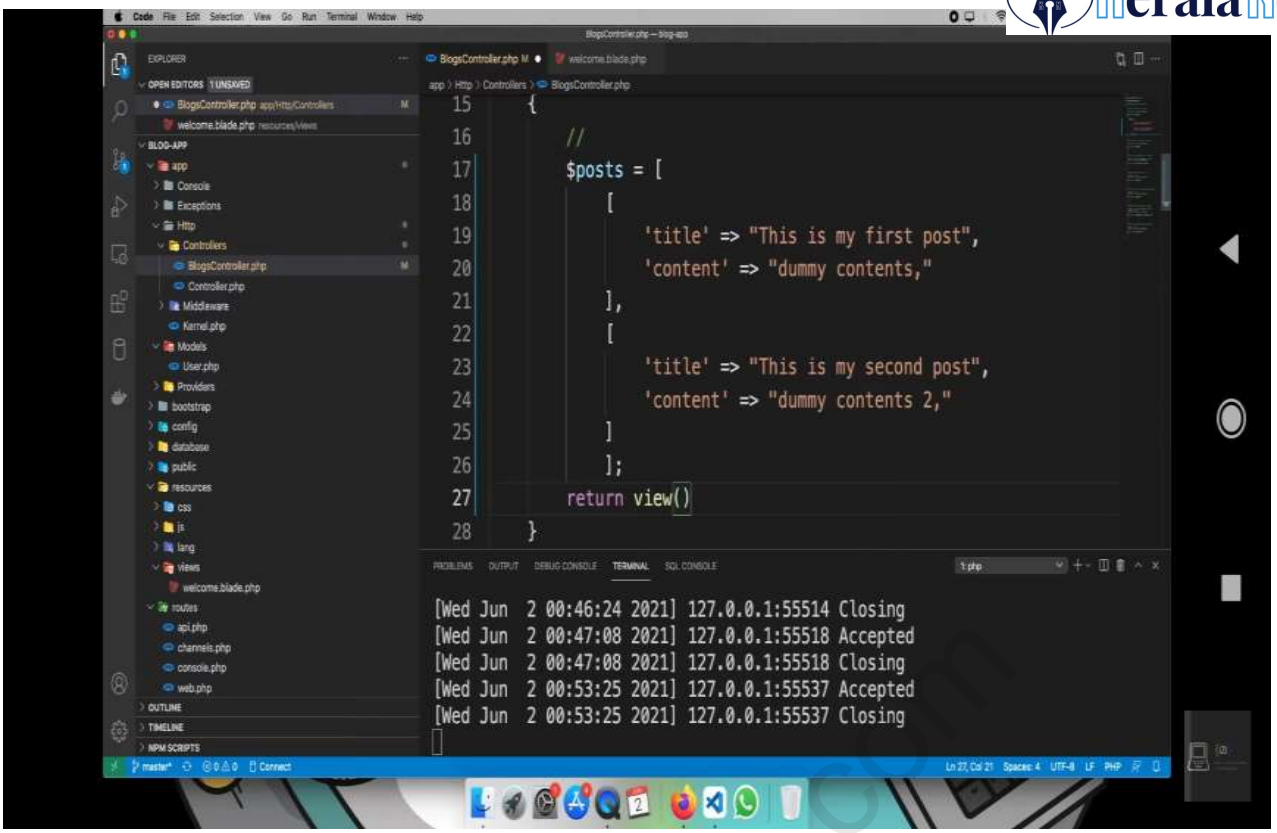


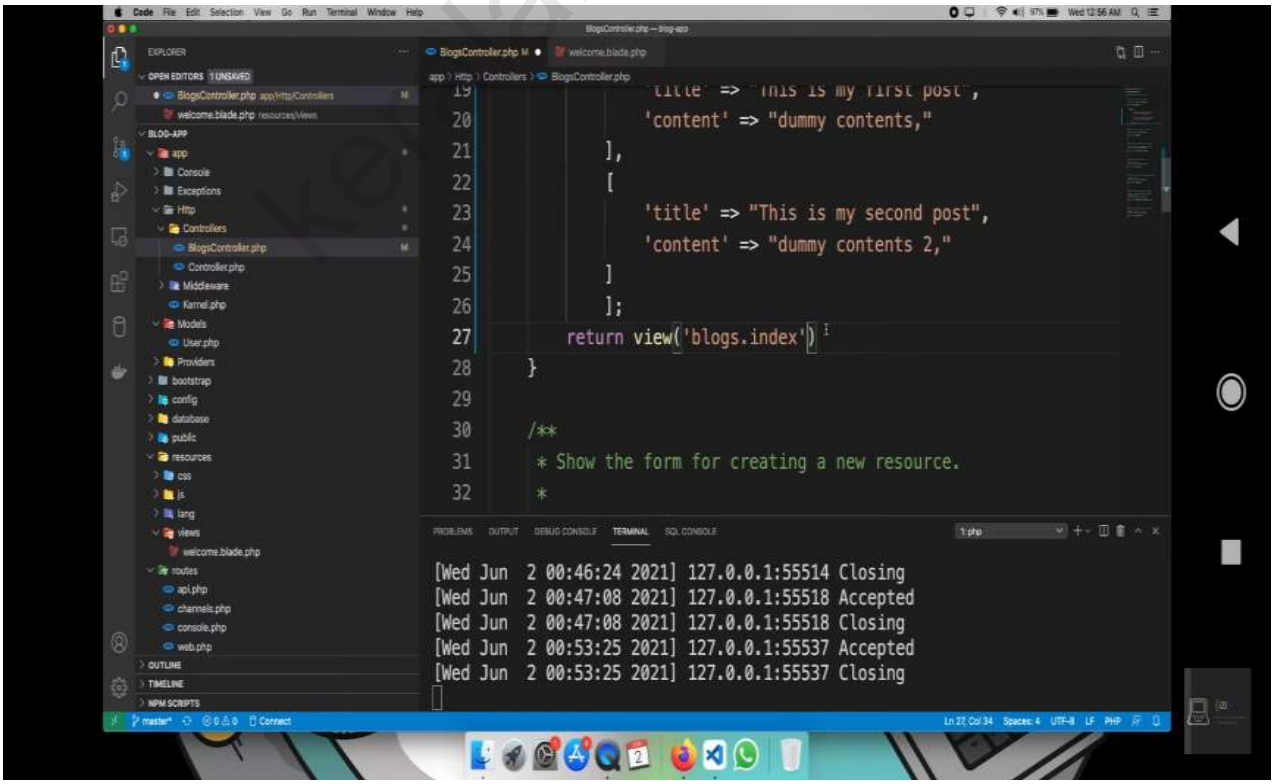


This is list of blogs!


BlogsController.php
app > Http > Controllers > BlogsController.php
8 {
9 /**
10 * Display a listing of the resource.
11 *
12 * @return \Illuminate\Http\Response
13 */
14 public function index()
15 {
16 //
17 return "This is list of blogs!";
18 }
19
20 /**
[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


BlogsController.php
welcome.blade.php
app > Http > Controllers > BlogsController.php
9 /**
10 * Display a listing of the resource.
11 *
12 * @return \Illuminate\Http\Response
13 */
14 public function index()
15 {
16 //
17 $posts = [
18 [
19 'title' => "This is my first post",
20 'content' => "dummy contents,"
21 ]
22 ]
[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


BlogsController.php
welcome.blade.php
app > Http > Controllers > BlogsController.php
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 }
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE
[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


BlogsController.php
welcome.blade.php
app > Http > Controllers > BlogsController.php
'content' => "dummy contents,"
],
[
'title' => "This is my second post",
'content' => "dummy contents 2,"
]
];
return view('blogs.index')
}
/**
* Show the form for creating a new resource.
*
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL SQL CONSOLE
[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