Module 5 (CO3, Cognitive Knowledge Level: Apply)

- Module- 5 (JSON, Laravel)
- 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.

EY

JSON Data Interchange Format

JSON

- JSON or JavaScript Object Notation is a lightweight text-based open standard designed for human-readable data interchange.
- Conventions used by JSON are known to programmers, which include C, C++, Java, Python, Perl, etc.
- JSON stands for JavaScript Object Notation.
- The format was specified by Douglas Crockford.
- It was designed for human-readable data interchange.
- It has been extended from the JavaScript scripting language.
- The filename extension is .json.
- JSON Internet Media type is application/json. The Uniform Type Identifier is public.json.

JSON-OVERVIEW

- JSON: JavaScript Object Notation.
- JSON is a syntax for storing and exchanging data.
- JSON is text, written with JavaScript object notation.
- JSON is a lightweight data-interchange format
- JSON is "self-describing" and easy to understand
- JSON is language independent
- JSON uses JavaScript syntax, but the JSON format is text only.
- Text can be read and used as a data format by any programming language.
- JavaScript has a built in function to convert a string, written in JSON format, into native JavaScript objects: JSON.parse()

JSON-Syntax

- The JSON syntax is a subset of the JavaScript syntax.
- JSON syntax is derived from JavaScript object syntax:
- Data is in name/value pairs .Data is separated by commas
- 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 supports the following two data structures:
  - Collection of name/value pairs:
    - This Data Structure is supported by different programming languages.
  - Ordered list of values
    - It includes array, list, vector or sequence etc.

JSON-Syntax

- The JSON format is almost identical to JavaScript objects.
- In JSON, *keys* must be strings, written with double quotes:
- The following example shows how to use JSON to store information related to books based on their topic and edition.
- Book array stores information about two books.
- Each book is having id, language, edition and author fields

```
{
    "book": [
    {
       "id":"01",
       "language": "Java",
       "edition": "third",
       "author": "Herbert Schildt"
    },
    {
       "id":"07",
       "language": "C++",
       "edition": "second"
       "author": "E.Balagurusamy"
    }]
}
```

JSON-Data

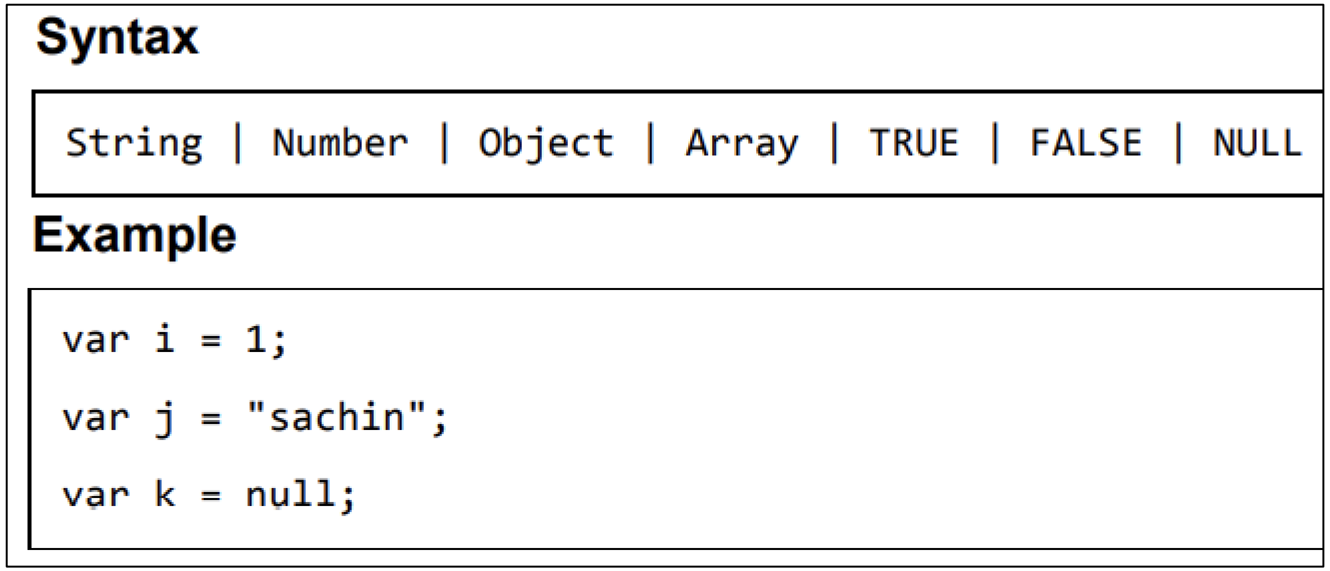
- JSON Data - A Name and a Value
- JSON data is written as name/value pairs.
- A name/value pair consists of a field name (in double quotes), followed by a colon, followed by a value:
- Example "name":"John“
- JSON names require double quotes. JavaScript names don’t.
- In JSON, keys must be strings, written with double quotes:

  JSON { "name":"John" }
- In JavaScript, keys can be strings, numbers, or identifier names:

  JavaScript { name:"John" }

JSON-Values

- In JSON,values must be one of the following data types: string,number,object (JSON object),array,Boolean,null



Syntax

```
String | Number | Object | Array | TRUE | FALSE | NULL
```

Example

```
var i = 1;
var j = "sachin";
var k = null;
```

- In JavaScript values can be all of the above, plus any other valid JavaScript expression, including: a function, a date , undefined
- In JSON,string values must be written with double quotes { "name":"John" }
- In JavaScript, you can write string values with double or single quotes:

  { name:'John' }

JSON-Values

- With JavaScript you can create an object and assign data to it:

```
var person = { "name":"John", "age":31, "city":"NewYork" };
```

- You can access a JavaScript object like this:

```
person.name; // returns John
OR
 person["name"];
```

- Data can be modified like this:

```
 person.name = "Gilbert";
 Or
 person["name"] = "Gilbert";
```

- The file type for JSON files is ".json"
- The MIME type for JSON text is "application/json"

JSON Data Types

JSON-Data Types

Type	Description
Number	double- precision floating-point format in JavaScript
String	double-quoted Unicode with backslash escaping
Boolean	true or false
Array	an ordered sequence of values
Value	it can be a string, a number, true or false, null etc
Object	an unordered collection of key:value pairs
Whitespace	can be used between any pair of tokens
null	empty

JSON-Data Types

- In JSON, values must be one of the following data types:
- a string
- a number
- an object (JSON object)
- an array
- a boolean
- null
- JSON values cannot be one of the following data types:
- a function
- a date
- undefined

JSON-Data Types

- **JSON Numbers**

Syntax

```
var json-object-name = {"string" : number_value, .......}
```

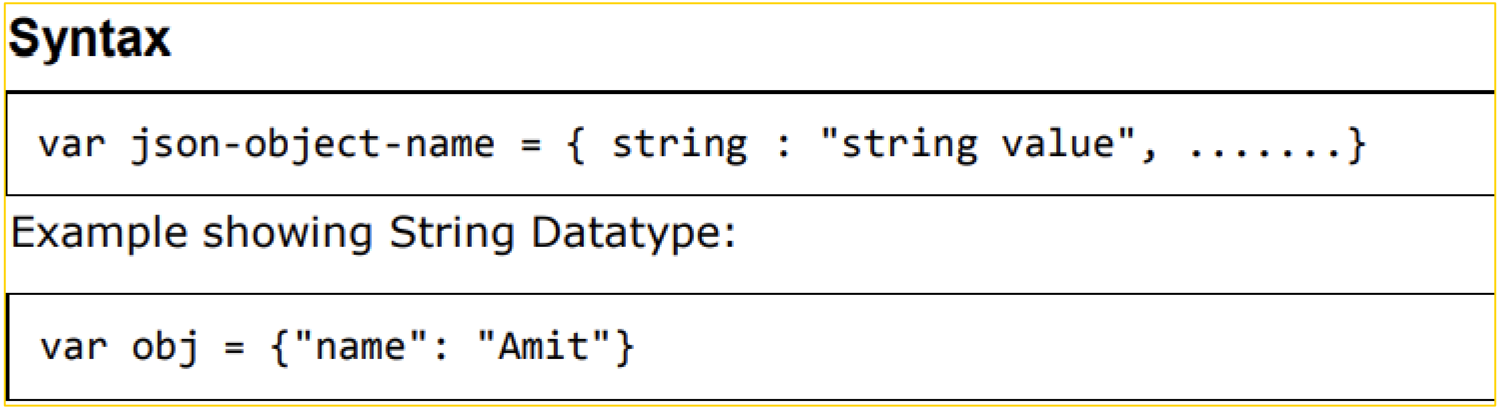
- Numbers in JSON must be an integer or a floating point.
- It is a double precision floating-point format in JavaScript and it depends on implementation.
- Octal and hexadecimal formats are not used. No NaN or Infinity is used in Number. { "age":30 }

```
var obj = {"marks": 97}
```

Integer	Digits 1-9, 0 and positive or negative
Fraction	Fractions like .3, .9
Exponent	Exponent like e, e+, e-,E, E+, E-

JSON-Data Types

- **JSON Strings**
- Strings in JSON must be written in double quotes. { "name":"John" }
- It is a sequence of zero or more double quoted Unicode characters with backslash escaping.
- Character is a single character string i.e. a string with length 1.



Syntax

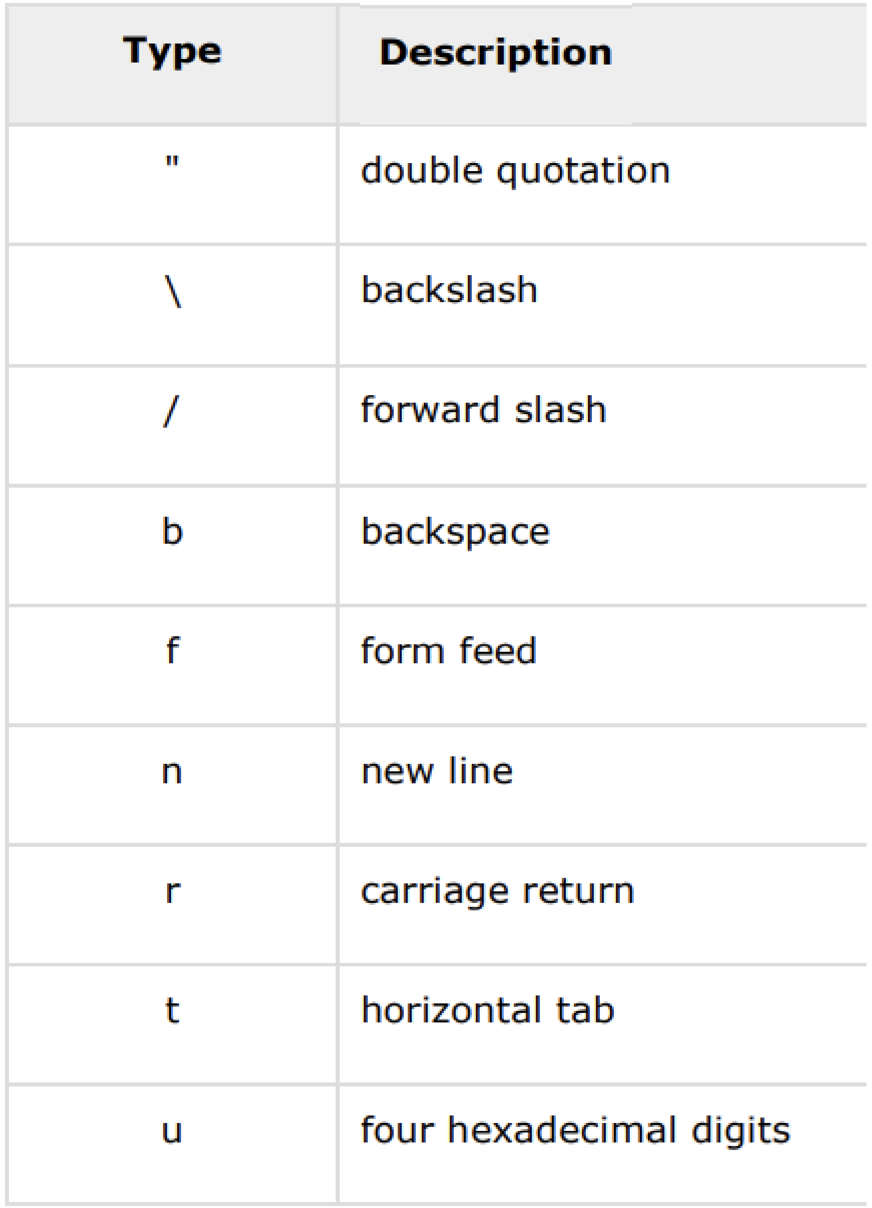
```
var json-object-name = { string : "string value", .......}
```

Example showing String Datatype:

```
var obj = {"name": "Amit"}
```

JSON-Data Types

- **JSON Strings**
- The table shows various special characters that you can use in strings of a JSON document
- A *string* is a sequence of zero or more Unicode characters, wrapped in double quotes, using backslash escapes.
- A character is represented as a single character string.
- A string is very much like a C or Java string.



Type	Description
"	double quotation
\	backslash
/	forward slash
b	backspace
f	form feed
n	new line
r	carriage return
t	horizontal tab
u	four hexadecimal digits

JSON-Data Types

- JSON Objects Values in JSON can be objects.

  { "employee":{ "name":"John", "age":30, "city":"NewYork" } }

- Objects as values in JSON must follow the same rules as JSON objects.
- It is an unordered set of name/value pairs. Objects are enclosed in curly braces that is, it starts with '{' and ends with '}'.
- Each name is followed by ':'(colon) and the key/value pairs are separated by , (comma).
- The keys must be strings and should be different from each other. Objects should be used when the key names are arbitrary strings.

Syntax

```
{ string : value, .......}
```

```
{
 "id": "011A",
 "language": "JAVA",
 "price": 500,
}
```

JSON-Data Types

- JSON Arrays
- It is an ordered collection of values.
- These are enclosed in square brackets which means that array begins with .[. and ends with .]..
- The values are separated by , (comma).
- Array indexing can be started at 0 or 1.
- Arrays should be used when the key names are sequential integers.
- Values in JSON can be arrays.

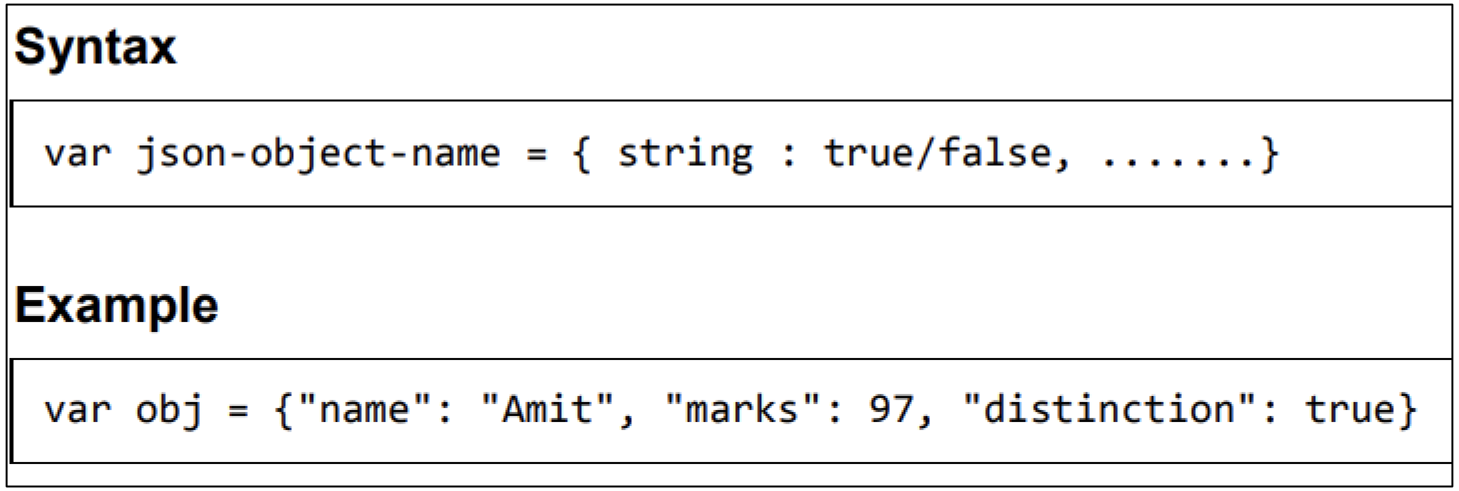
{ "employees":["John", "Anna", "Peter"] }

```
{"books": [
   { "language":"Java" , "edition":"second" },
   { "language":"C++" , "lastName":"fifth" },
   { "language":"C" , "lastName":"third" }
 ] }
```

JSON-Data Types

- JSON Booleans Values in JSON can be true/false.

  { "sale":true }



Syntax

```
var json-object-name = { string : true/false, .......}
```

Example

```
var obj = {"name": "Amit", "marks": 97, "distinction": true}
```

- JSON null Values in JSON can be null. It means empty type.

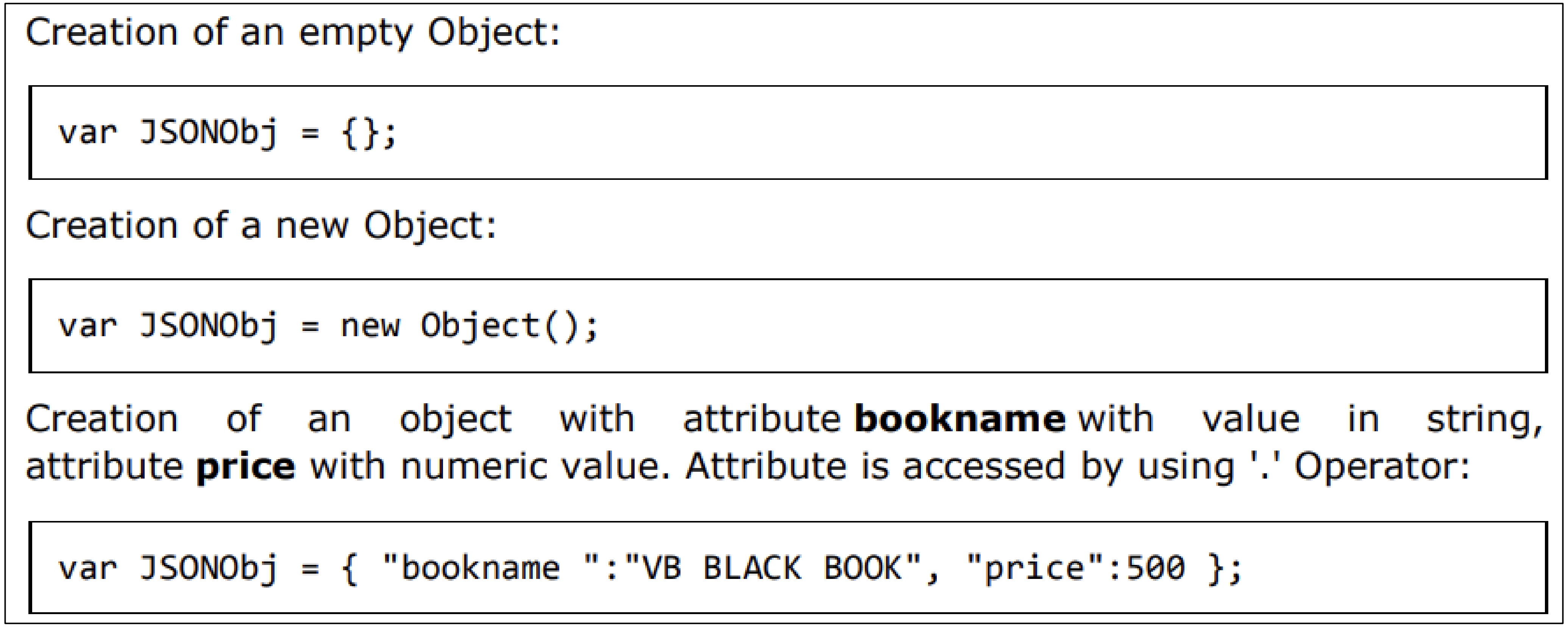
  { "middlename":null }

- Whitespace:It can be inserted between any pair of tokens. It can be added to make a code more readable. Example shows declaration with and without whitespace:

JSON OBJECT

JSON-Objects

- Creating Simple Objects
- JSON objects can be created with JavaScript.



Creation of an empty Object:

```
var JSONObj = {};
```

Creation of a new Object:

```
var JSONObj = new Object();
```

Creation of an object with attribute **bookname** with value in string, attribute **price** with numeric value. Attribute is accessed by using '.' Operator:

```
var JSONObj = { "bookname ":"VB BLACK BOOK", "price":500 };
```

Accessing JSON Objects

- You can access the object values by using dot (.) notation:

```
myObj = { "name":"John", "age":30, "car":null };
 x = myObj.name;
```

- You can also access the object values by using bracket ([]) notation:

```
myObj = { "name":"John", "age":30, "car":null };
 x = myObj["name"];
```

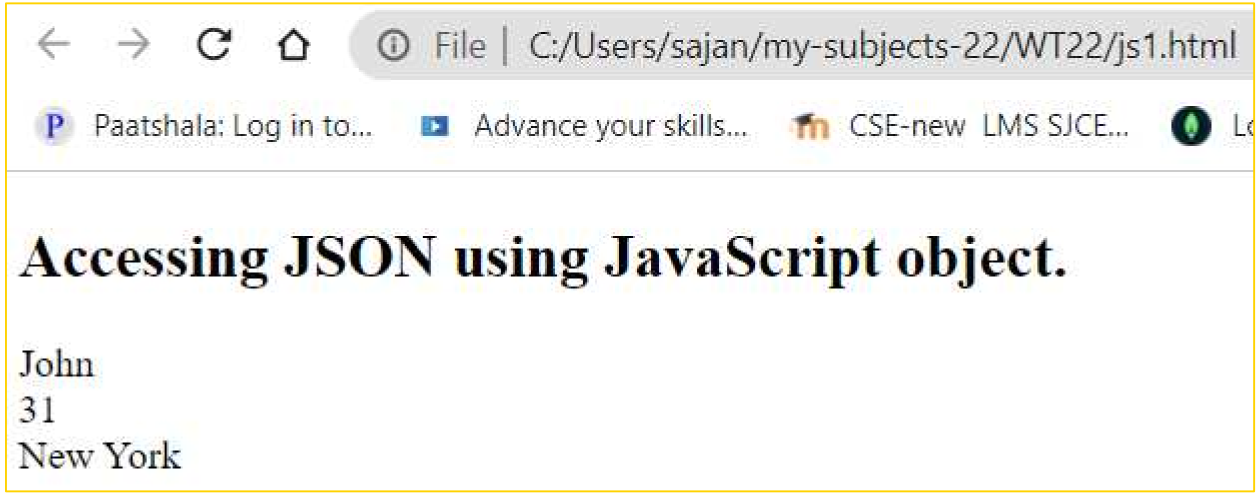
- You can loop through object properties by using the for-in loop:

```
person = { "name":"John", "age":30, "car":null };
for (x in person)
 {
 alert(x+"="+person[x])
 }
```

```
o/p:
Name=john
Age=30
Car=null
```

Accessing JSON Object in JavaScript

```
<!DOCTYPE html>
<html><body>
<h2>Accessing JSON using JavaScript
</h2>
<p id="demo"></p>
<script>
var myObj = { "name":"John", "age":31, "city":"New York" };
document.getElementById("demo").innerHTML = myObj.name+"<br/>"+myObj.age+"<br/>"+myObj.city;
</script>
</body></html>
```

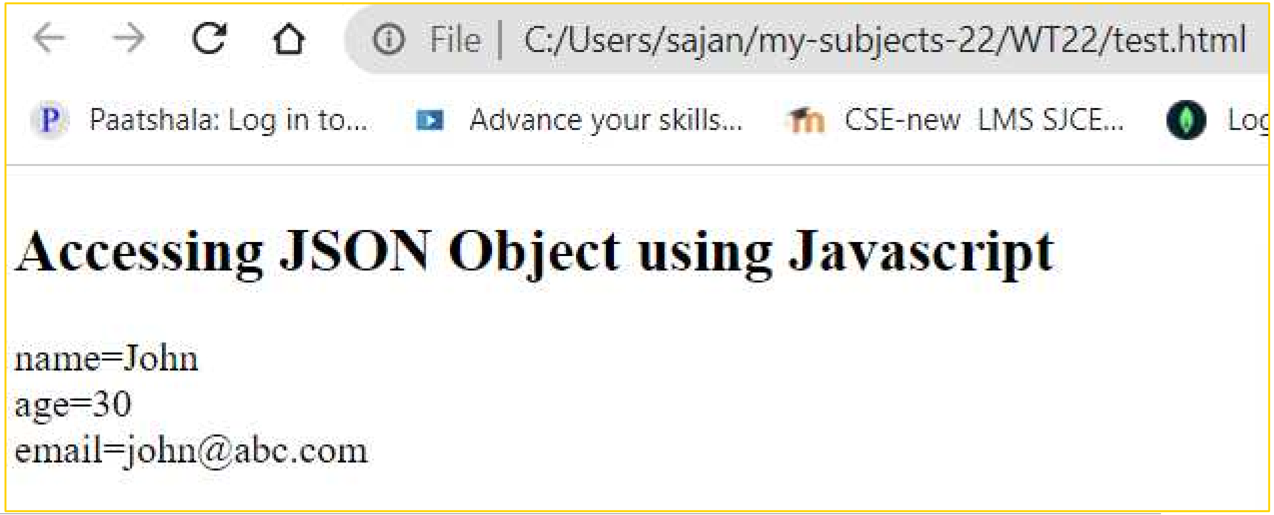


Accessing JSON using JavaScript object.

```
John
31
New York
```

Accessing JSON Object in Javascript

```
<!DOCTYPE html>
<html><body>
<h2>Accessing JSON Object using Javascript
</h2>
<script>
var myObj = { "name":"John", "age":30, "email": "john@abc.com" };
for (x in myObj)
         {

        document.write(x+"="+myObj[x]+"<br/>");
         }

</script>
</body></html>
```



Accessing JSON Object using Javascript

```
name=John
age=30
email=john@abc.com
```

Nested JSON Objects

- Values in a JSON object can be another JSON object.

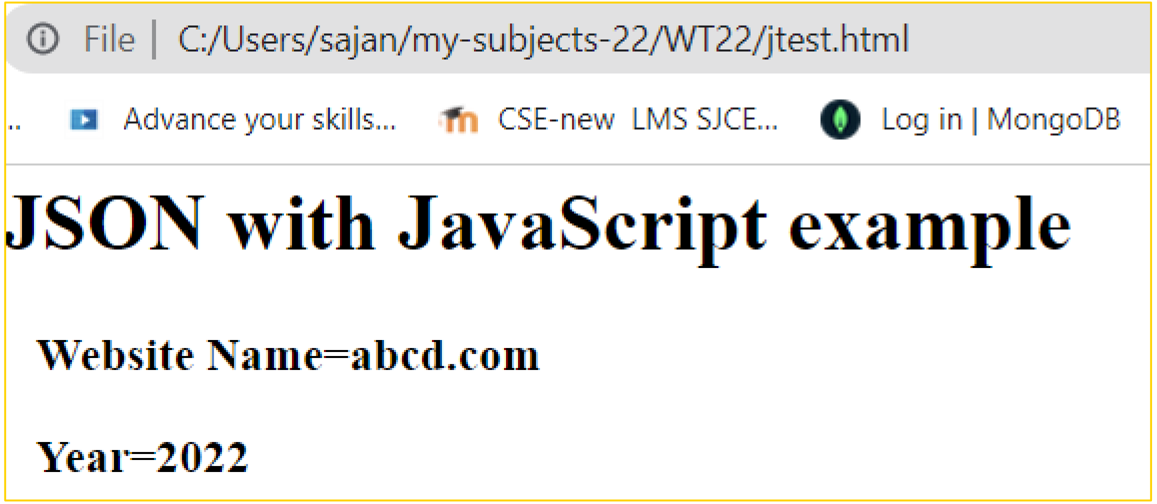
```
      myObj = { "name":"John”, "age":30,
             "cars": { "car1":"Ford", "car2":"BMW", "car3":"Fiat" }
               }
```

- You can access nested JSON objects by using the dot notation or bracket notation: Example

```
x = myObj.cars.car2;
     or
x = myObj.cars["car2"];
```

creation of an object in JavaScript using JSON,

```
<html>
<head>
<title>Creating Object JSON with JavaScript</title>
<script language="javascript" >
 var JSONObj = { "name" : "abcd.com", "year" : 2022 };
 document.write("<h1>JSON with JavaScript example</h1>");
 document.write("<br>");
 document.write("<h3>Website Name="+JSONObj.name+"</h3>");
 document.write("<h3>Year="+JSONObj.year+"</h3>");
</script>
</head>
<body>
</body>
</html>
```



```
JSON with JavaScript example

Website Name=abcd.com

Year=2022
```

JSON SCHEMA

JSON Schema

- JSON Schema is a specification for JSON based format for defining the structure of JSON data.
- It was written under IETF draft which expired in 2011.
- JSON Schema , describes your existing data format.
- JSON Schema is a powerful tool for validating the structure of JSON data.
- Complete structural validation, useful for automated testing, validating client-submitted data.
- There are several JSON Schema Validation Libraries and validators currently available for different programming languages.
- JSON Schema itself is written in JSON.
- It is data itself, not a computer program. It's just a declarative format for "describing the structure of other data".

JSON Schema

- For example, you could imagine representing information about a person in JSON in different ways:
- Both representations are equally valid, though one is clearly more formal than the other
- Suppose we need to know what fields are expected, and how the values are represented. Means we want to validate the data
- That's where JSON Schema comes in.

```
{
  "name": "George Washington",
  "birthday": "February 22, 1732",
  "address": "Mount Vernon, Virginia, United States"
}

{
  "first_name": "George",
  "last_name": "Washington",
  "birthday": "1732-02-22",
  "address": {
    "street_address": "3200 Mount Vernon Memorial Highway",
    "city": "Mount Vernon",
    "state": "Virginia",
    "country": "United States"
  }
}
```

JSON Schema

```
{
  "type": "object",
  "properties": {
    "first_name": { "type": "string" },
    "last_name": { "type": "string" },
    "birthday": { "type": "string", "format": "date" },
    "address": {
      "type": "object",
      "properties": {
        "street_address": { "type": "string" },
        "city": { "type": "string" },
        "state": { "type": "string" },
        "country": { "type" : "string" }
      }
    }
  }
}
```

```
{
  "name": "George Washington",
  "birthday": "February 22, 1732",
  "address": "Mount Vernon, Virginia, United States"
}
```

✗

By “validating” the first example against this schema, you can see that it fails. However, the second example passes:

```
{
  "first_name": "George",
  "last_name": "Washington",
  "birthday": "1732-02-22",
  "address": {
    "street_address": "3200 Mount Vernon Memorial Highway",
    "city": "Mount Vernon",
    "state": "Virginia",
    "country": "United States"
  }
}
```

✓

JSON Schema

- **Structure of a JSON Schema:** Since JSON format contains an object, array, and name-value pair elements. Name-value pairs are used for providing schema processing elements as well as validating the JSON content.
- **$schema"** To specify the version of JSON schema.
- **title and description:** To provide information about the schema.
- **required:** It's an array of elements that indicates which elements should be present.
- **additionalProperties:** To indicate whether existence of specified elements are allowed or not.
- **JSON content definition:**
- When a JSON object is defined, JSON schema uses name-value pair **"type":"object"**
- When arrays are defined, JSON schema uses the name-value pair **"type":"array"**

```
{  "$schema": "http://json-schema.org/draft-04/schema#",
   "title": "Product",
   "description": "A product from Acme's catalog",
   "type": "object",
   "properties": {
       "id": {
           "description": "The unique identifier for a product",
           "type": "integer"
       },
       "name": {
           "description": "Name of the product",
           "type": "string"
       },
       "price": {
           "type": "number",
           "minimum": 0,
           "exclusiveMinimum": true
       }
   },  "required": ["id", "name", "price"] }
```

Keywords	Description
$schema	The $schema keyword states that this schema is written according to the draft v4 specification.
title	You will use this to give a title to your schema.
description	A little description of the schema.
type	The type keyword defines the first constraint on our JSON data: it has to be a JSON Object.
properties	Defines various keys and their value types, minimum and maximum values to be used in JSON file.
required	This keeps a list of required properties.
minimum	This is the constraint to be put on the value and represents minimum acceptable value.
exclusiveMinimum	If "exclusiveMinimum" is present and has boolean value true, the instance is valid if it is strictly greater than the value of "minimum".

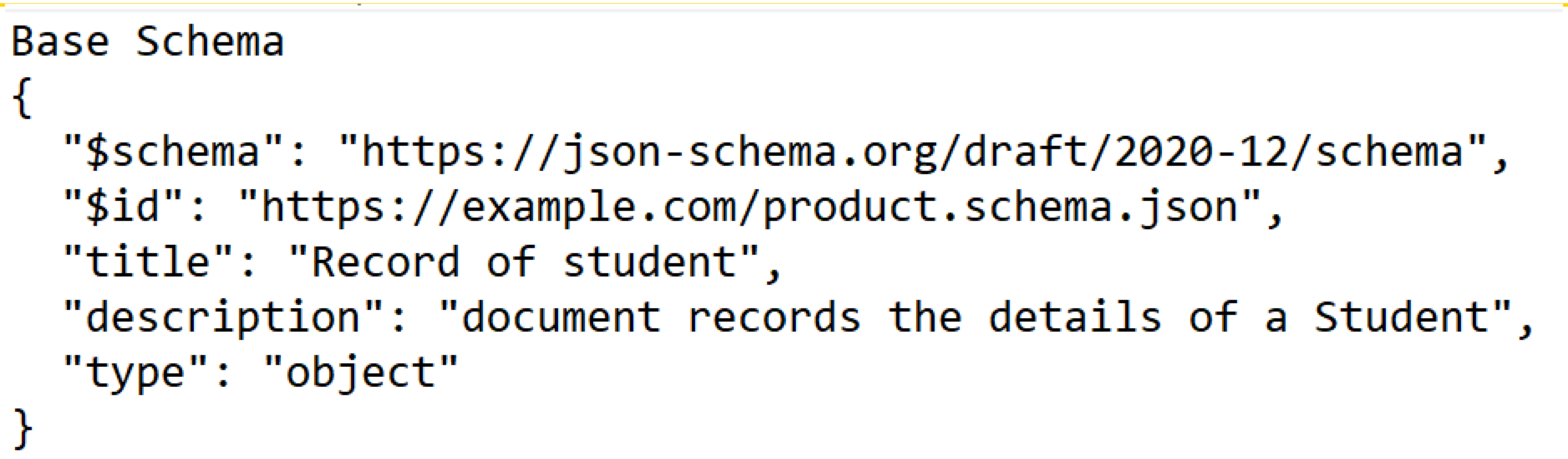
maximum	This is the constraint to be put on the value and represents maximum acceptable value.
exclusiveMaximum	If "exclusiveMaximum" is present and has boolean value true, the instance is valid if it is strictly lower than the value of "maximum".
multipleOf	A numeric instance is valid against "multipleOf" if the result of the division of the instance by this keyword's value is an integer.
maxLength	The length of a string instance is defined as the maximum number of its characters.
minLength	The length of a string instance is defined as the minimum number of its characters.
pattern	A string instance is considered valid if the regular expression matches the instance successfully.

JSON Schema

- Check the http://json-schema.org for the complete list of keywords that can be used in defining a JSON schema.
- The above schema can be used to test the validity of the following JSON code:

```
[
    {
        "id": 2,
        "name": "An ice sculpture",
        "price": 12.50,
    },
    {
        "id": 3,
        "name": "A blue mouse",
        "price": 25.50,
    }
]
```

document definition of a ‘Student document ‘using JSON Schema



```
Base Schema
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/product.schema.json",
  "title": "Record of student",
  "description": "document records the details of a Student",
  "type": "object"
}
```

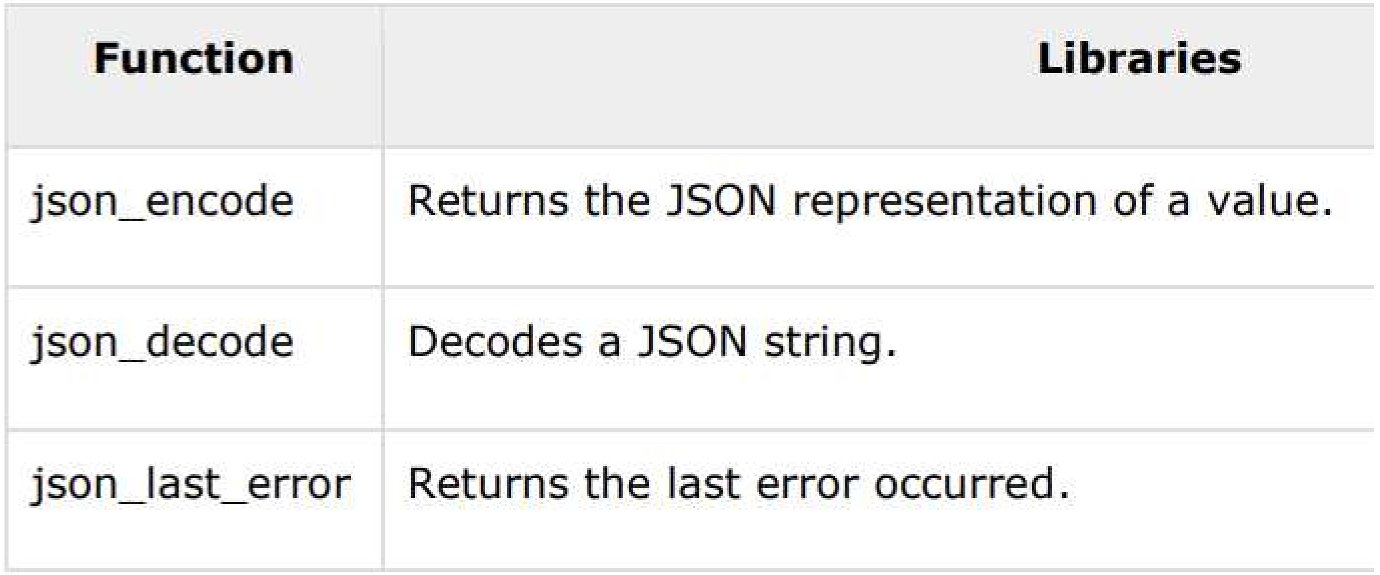
- $schema links to the resource that identifies the valid schemas
- $id keyword identifies the schema resource.
- The URI in this keyword is an identifier and not necessarily a network locator.
- It must represent a valid URI reference that is normalized and resolves absolute URI.
- It must not contain a non empty fragment . It should not contain an empty fragment.
- title keyword gives a short description of the schema
- description keyword to explain more about the schema

```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "https://example.com/product.schema.json",
  "title": "Record of student",
  "description": "document records the details of a Student",
  "type": "object",
    "properties": {
        "id": {
            "description": "A unique identifier for a Student",
            "type": "number"
              }
        "name": {
            "description": "full name of the student",
            "type": "string"
            "minLength":3
          },
        "age": {
            "description": "age of the student",
            "type": "number",
            "minimum": 16
        }
    }, "required": ["id","name","age"]
}
```

Manipulating JSON data with PHP Web

JSON – JSON with PHP

- As of PHP 5.2.0, the JSON extension is bundled and compiled into PHP by default.
- JSON Functions
- PHP json_encode() function is used for encoding JSON in PHP. This function returns the JSON representation of a value on success or FALSE on failure.
- PHP json_decode() function is used for decoding JSON in PHP. This function returns the value decoded from json to appropriate PHP type.



Function	Libraries
json_encode	Returns the JSON representation of a value.
json_decode	Decodes a JSON string.
json_last_error	Returns the last error occurred.

Encoding JSON in PHP (json_encode)

- As of PHP 5.2.0, the JSON extension is bundled and compiled into PHP by default.
- JSON Functions
- PHP json_encode() function is used for encoding JSON in PHP. This function returns the JSON representation of a value on success or FALSE on failure.
- PHP json_decode() function is used for decoding JSON in PHP. This function returns the value decoded from json to appropriate PHP type.

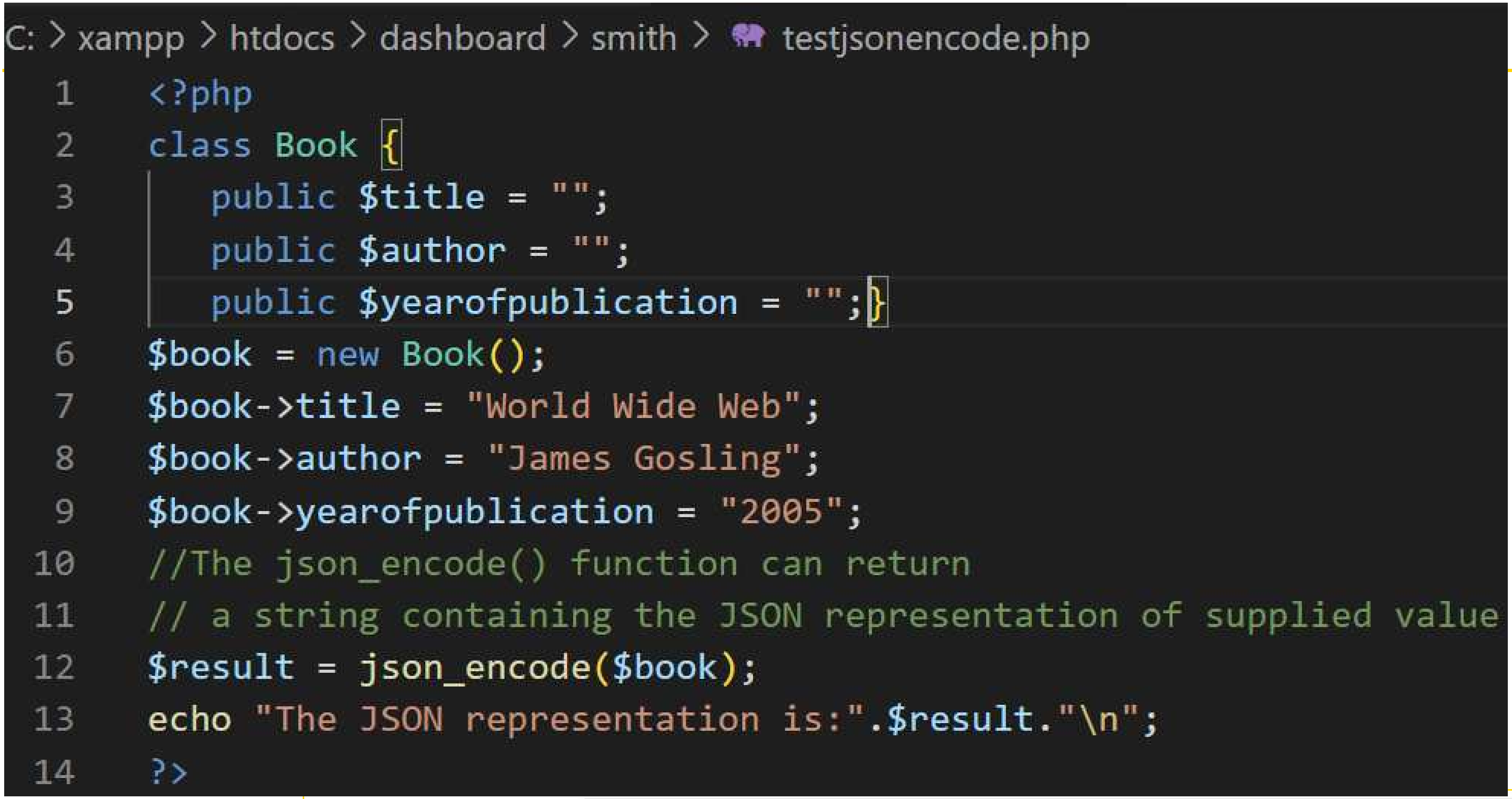
Syntax

```
string json_encode ( $value [, $options = 0 ] )
```

Parameters

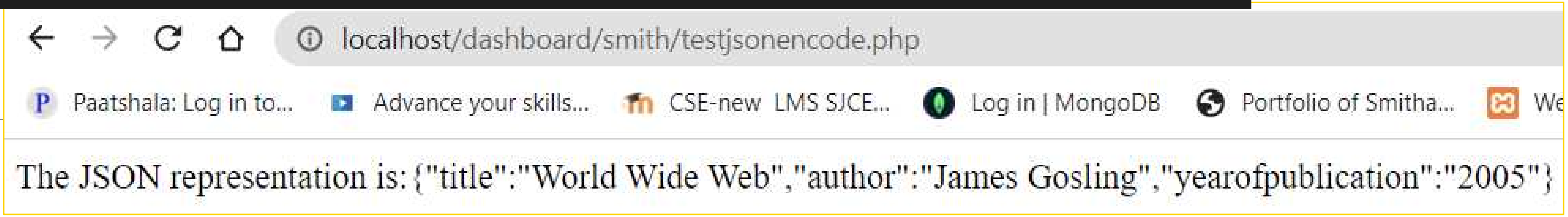
- **value**: The value being encoded. This function only works with UTF-8 encoded data.
- **options**: This optional value is a bitmask consisting of JSON_HEX_QUOT, JSON_HEX_TAG, JSON_HEX_AMP, JSON_HEX_APOS, JSON_NUMERIC_CHECK,JSON_PRETTY_PRINT, JSON_UNESCAPED_SLASHES, JSON_FORCE_OBJECT.

How the PHP objects can be converted into JSON?(json_encode)



C: > xampp > htdocs > dashboard > smith > testjsonencode.php

```php
<?php
class Book {
    public $title = "";
    public $author = "";
    public $yearofpublication = "";}
$book = new Book();
$book->title = "World Wide Web";
$book->author = "James Gosling";
$book->yearofpublication = "2005";
//The json_encode() function can return
// a string containing the JSON representation of supplied value
$result = json_encode($book);
echo "The JSON representation is:".$result."\n";
?>
```



localhost/dashboard/smith/testjsonencode.php

```
The JSON representation is:{"title":"World Wide Web","author":"James Gosling","yearofpublication":"2005"}
```

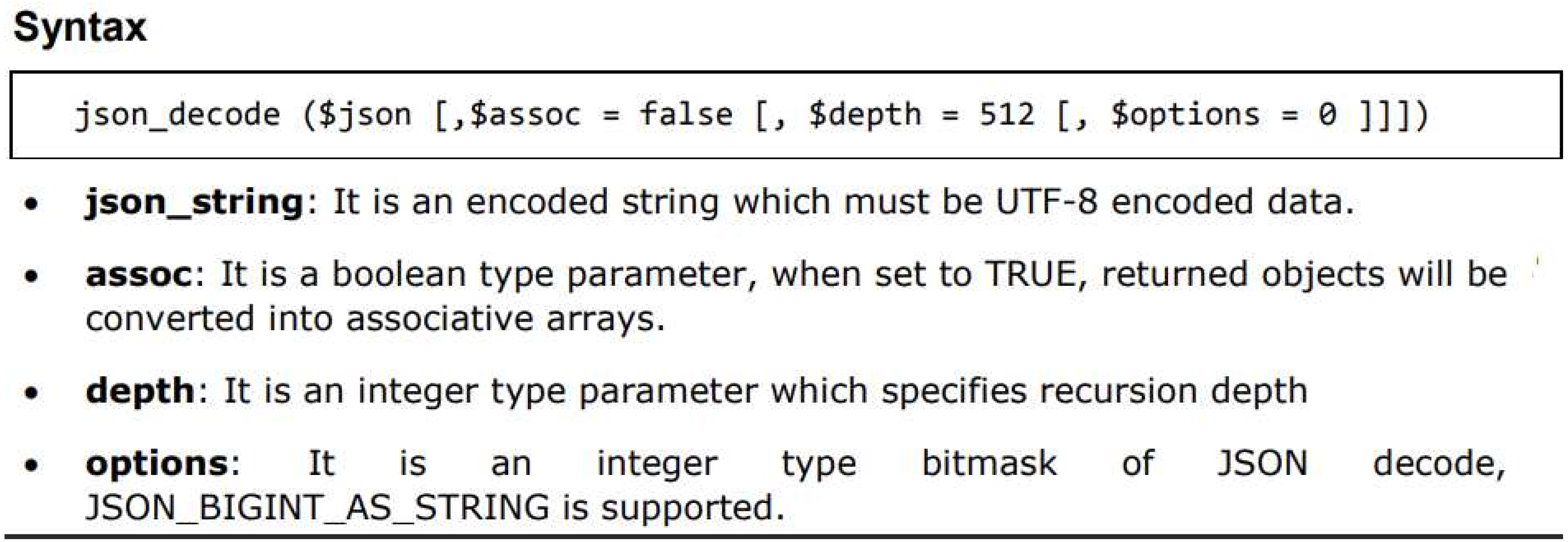
how the PHP objects can be converted into JSON?(json_encode)

```
<?php
   class Emp {
       public $name = "";
       public $hobbies  = "";
       public $birthdate = "";
   }
   $e = new Emp();
   $e->name = "sachin";
   $e->hobbies  = "sports";
   $e->birthdate = date('m/d/Y h:i:s a', "8/5/1974 12:20:03 p");
   $e->birthdate = date('m/d/Y h:i:s a', strtotime("8/5/1974 12:20:03"));
    echo json_encode($e);
 ?>
```

```
{"name":"sachin","hobbies":"sports","birthdate":"08\/05\/1974 12:20:03 pm"}
```

Decoding JSON in PHP (json_decode)

- PHP json_decode() function is used for decoding JSON in PHP. This function returns the value decoded from json to appropriate PHP type.



Syntax

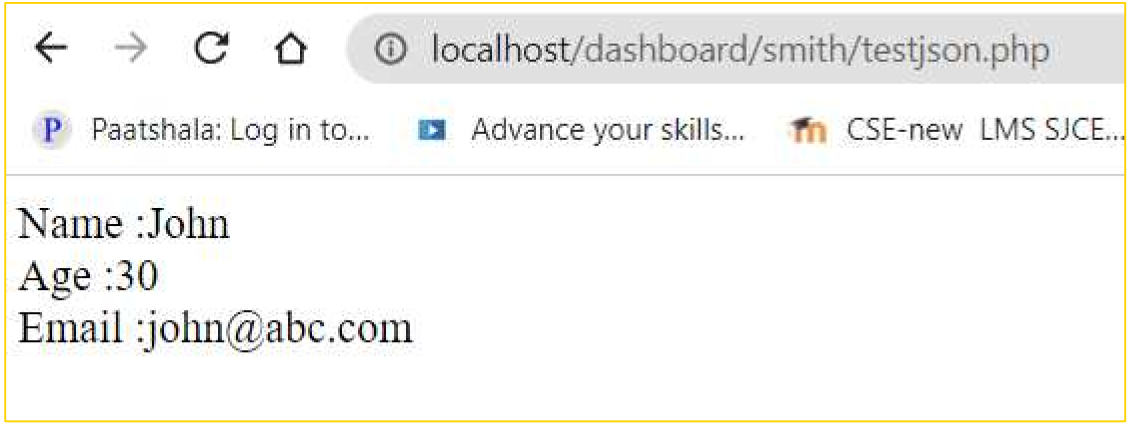
```
json_decode ($json [,$assoc = false [, $depth = 512 [, $options = 0 ]]])
```

- **json_string**: It is an encoded string which must be UTF-8 encoded data.
- **assoc**: It is a boolean type parameter, when set to TRUE, returned objects will be converted into associative arrays.
- **depth**: It is an integer type parameter which specifies recursion depth
- **options**: It is an integer type bitmask of JSON decode, JSON_BIGINT_AS_STRING is supported.

Decoding JSON in PHP (json_decode)

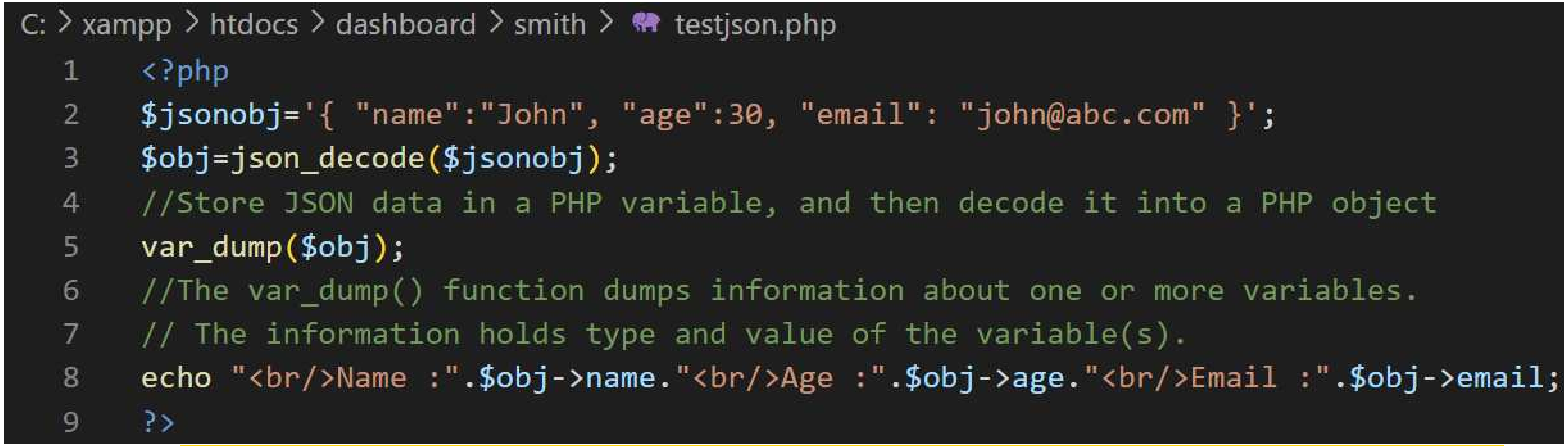
- PHP json_decode() function is used for decoding JSON in PHP. This function returns the value decoded from json to appropriate PHP type.

```
<?php
$jsonobj='{ "name":"John", "age":30, "email": "john@abc.com" }';
$obj=json_decode($jsonobj);
echo "Name :".$obj->name."<br/>Age :".$obj->age."<br/>Email :".$obj->email;
?>
```

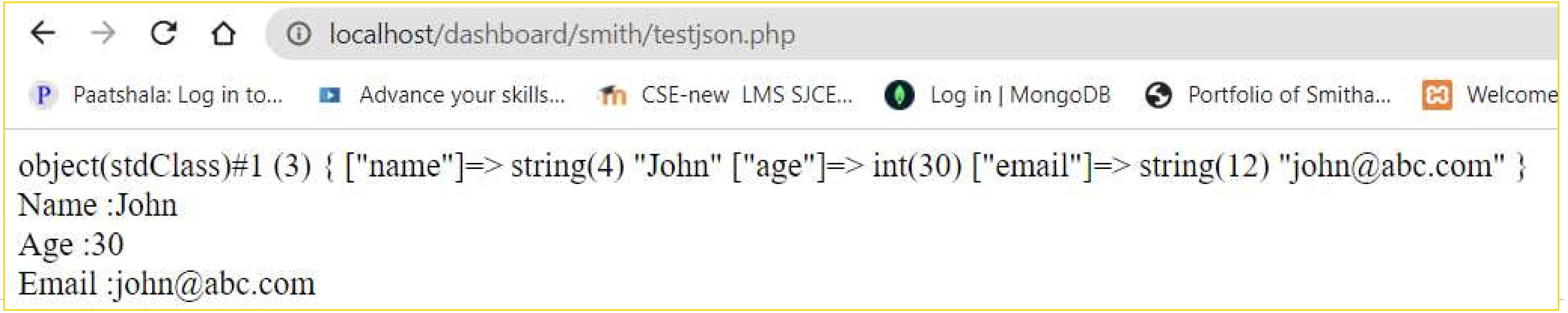


```
Name :John
Age :30
Email :john@abc.com
```

Decoding JSON in PHP (json_decode)


C: > xampp > htdocs > dashboard > smith > testjson.php

```php
<?php
$jsonobj='{ "name":"John", "age":30, "email": "john@abc.com" }';
$obj=json_decode($jsonobj);
//Store JSON data in a PHP variable, and then decode it into a PHP object
var_dump($obj);
//The var_dump() function dumps information about one or more variables.
// The information holds type and value of the variable(s).
echo "<br/>Name :".$obj->name."<br/>Age :".$obj->age."<br/>Email :".$obj->email;
?>
```



```
object(stdClass)#1 (3) { ["name"]=> string(4) "John" ["age"]=> int(30) ["email"]=> string(12) "john@abc.com" }
Name :John
Age :30
Email :john@abc.com
```

Development Framework , Laravel

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.

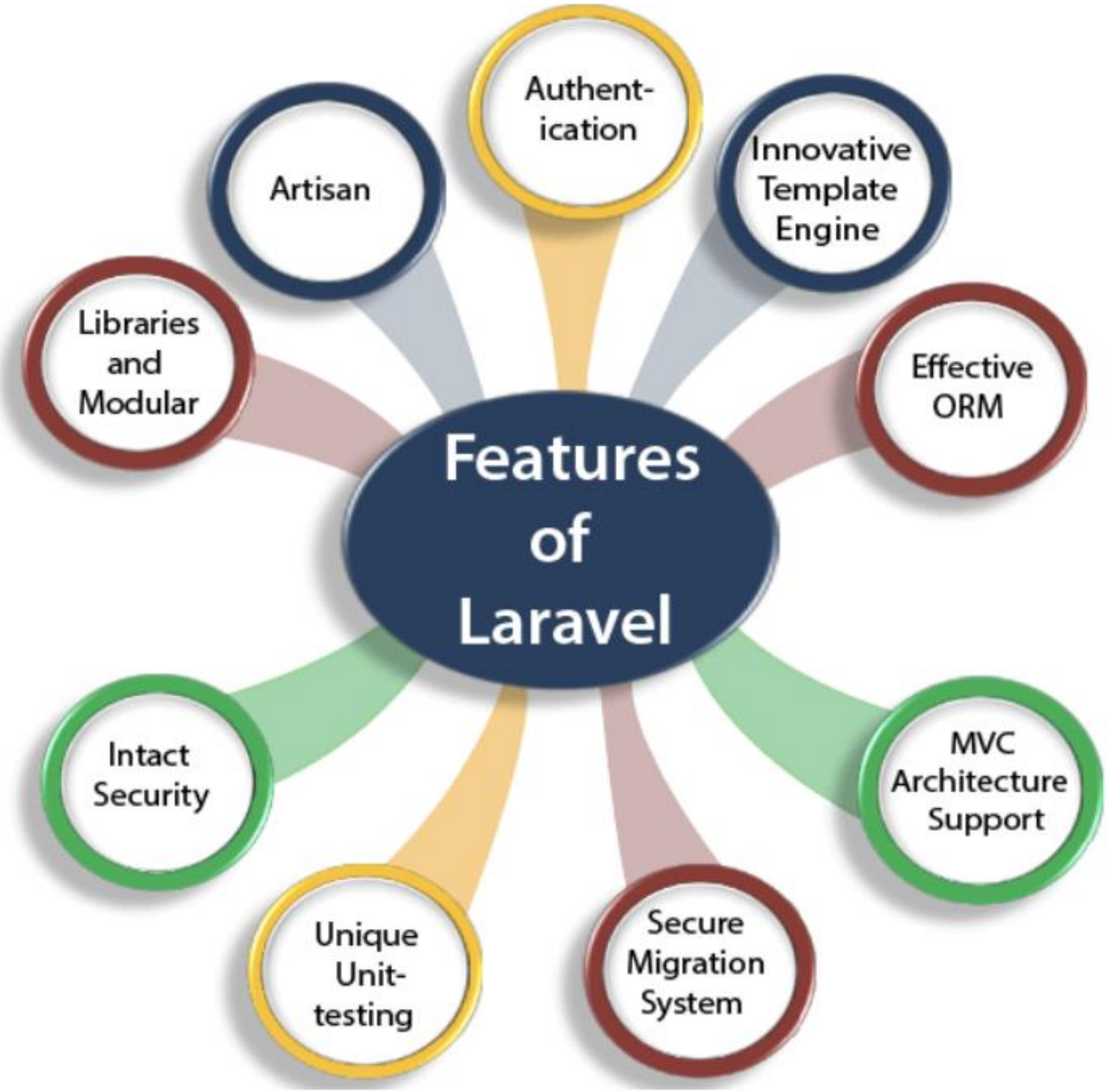
Web Development Frameworks:Laravel

- Frameworks like Laravel , Symfony, Silex, Lumen, and Slim — prepackage a collection of third-party components together with custom framework "glue" like configuration files, service providers, prescribed directory structures, and application bootstraps.
- Laravel's architecture is different from other PHP frameworks. This framework works on an MVC-based design.
- Laravel uses one of the powerful default template engines i.e. Blade template engine. It allows Laravel developers to use plain PHP code. Using this template engine, create an app without additional costs.
- Laravel supports only these database languages
  - MySQL
  - PostgreSQL
  - SQLite, and
  - SQLServer

Web Development Frameworks : Laravel

- Laravel is a web application framework with expressive, elegant syntax.
- Laravel is the best choice for building modern, full-stack web applications.
- A Progressive Framework
  - Laravel grows with you. Laravel gives you robust tools for dependency injection, unit testing, queues, real-time events, and more.
- A Scalable Framework
  - Laravel is incredibly scalable . Laravel's built-in support for fast, distributed cache systems , horizontal scaling with easily scaled to handle hundreds of millions of requests per month.
- A Community Framework
  - Laravel combines the best packages in the PHP ecosystem to offer the most robust and developer friendly framework available.

Features of Laravel

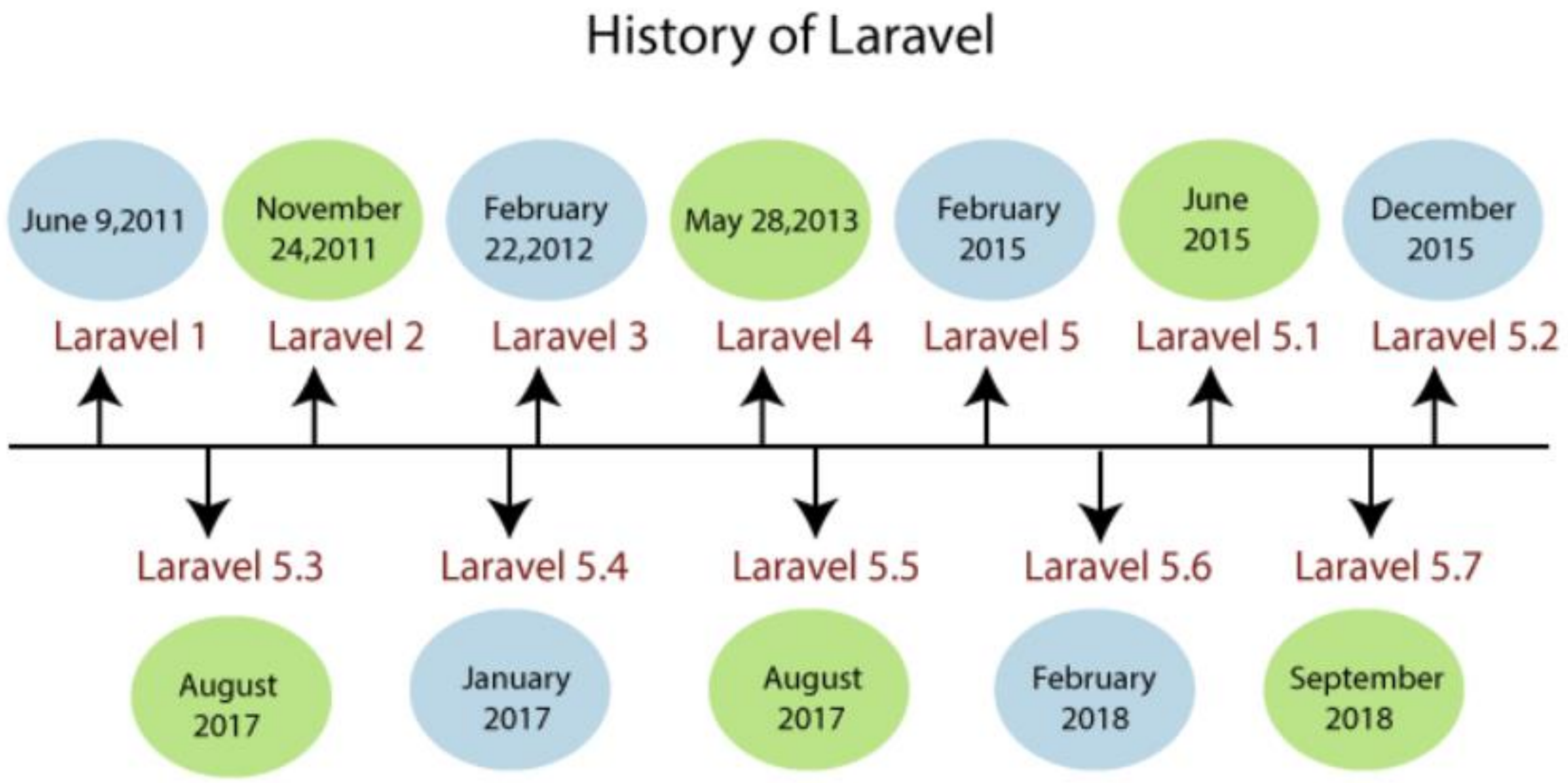

Features of Laravel
Authentication
Innovative Template Engine
Effective ORM
MVC Architecture Support
Secure Migration System
Unique Unit-testing
Intact Security
Libraries and Modular
Artisan

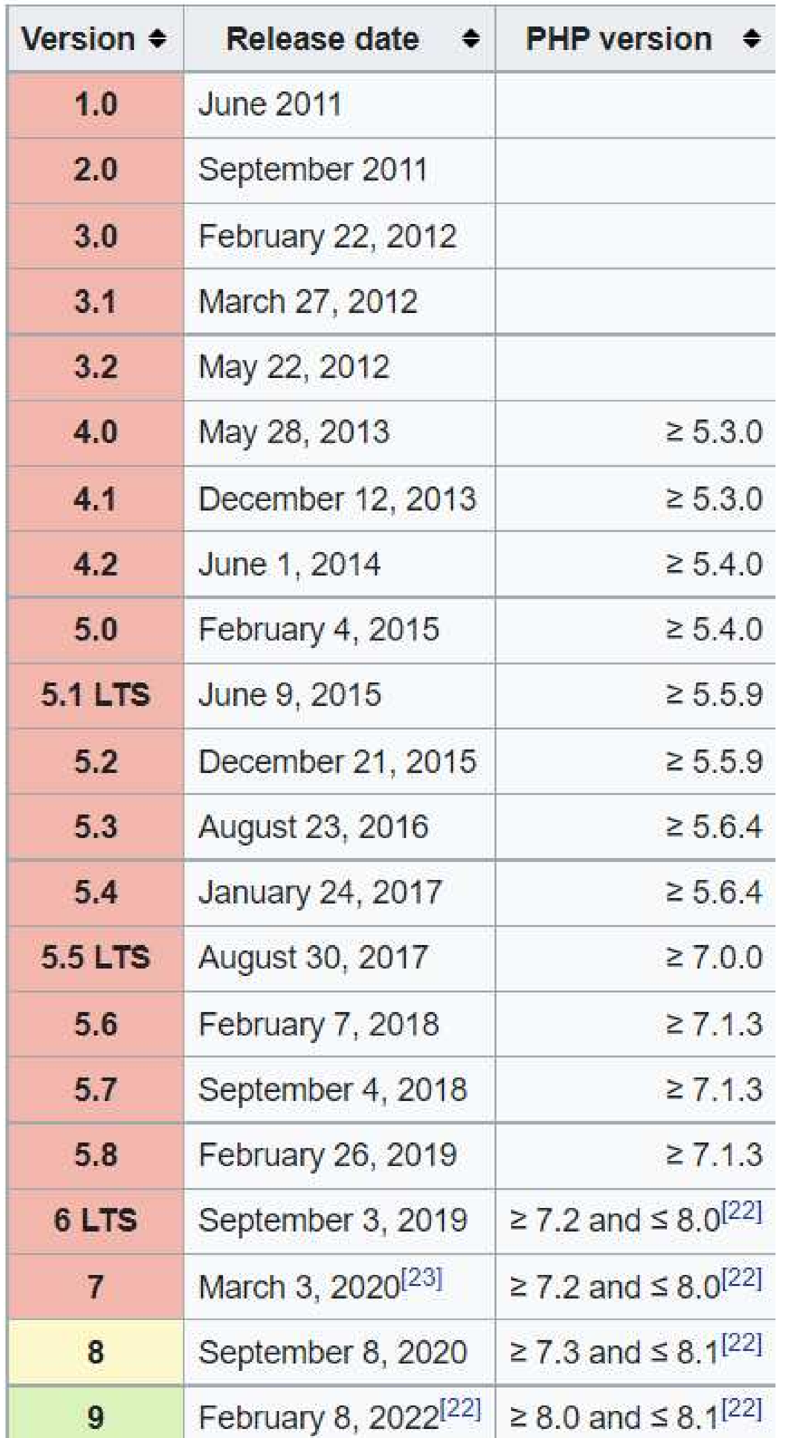
Features of Laravel

- Authentication
  - Laravel contains an inbuilt authentication system, you only need to configure models, views, and controllers to make the application work.
- Innovative Template Engine
  - Laravel provides an innovative template engine which allows the developers to create solid structures for an application.
- Effective ORM
  - Laravel contains an inbuilt ORM , allows the developers to query the database tables by using the simple PHP syntax without writing any SQL code.
- MVC Architecture Support
  - Laravel supports MVC architecture. It provides faster development process as in MVC,and it separates the business logic from the presentation logic.
- Secure Migration System
  - **Laravel framework** can expand the database and the migration process of Laravel is very secure and full-proof. In the whole process, **php code** is used rather than **SQL code**.

Features of Laravel

- **Unique Unit-testing**
- Laravel provides a unique unit-testing. In Laravel, developers can also write the test cases in their own code.
- **Intact Security**
- Laravel has an inbuilt web application security, i.e., it itself takes care of the security of an application. It uses "Bcrypt Hashing Algorithm" to generate the salted password
- **Libraries and Modular**
- Laravel is very popular as some Object-oriented libraries, and pre-installed libraries are added in this framework, these pre-installed libraries are not added in other **php frameworks**. This framework is divided into several modules that follow the php principles allowing the developers to build responsive and modular apps.
- **Artisan**
- Laravel framework provides a built-in tool for a command-line known as **Artisan** that performs the repetitive programming tasks that do not allow the php developers to perform manually.





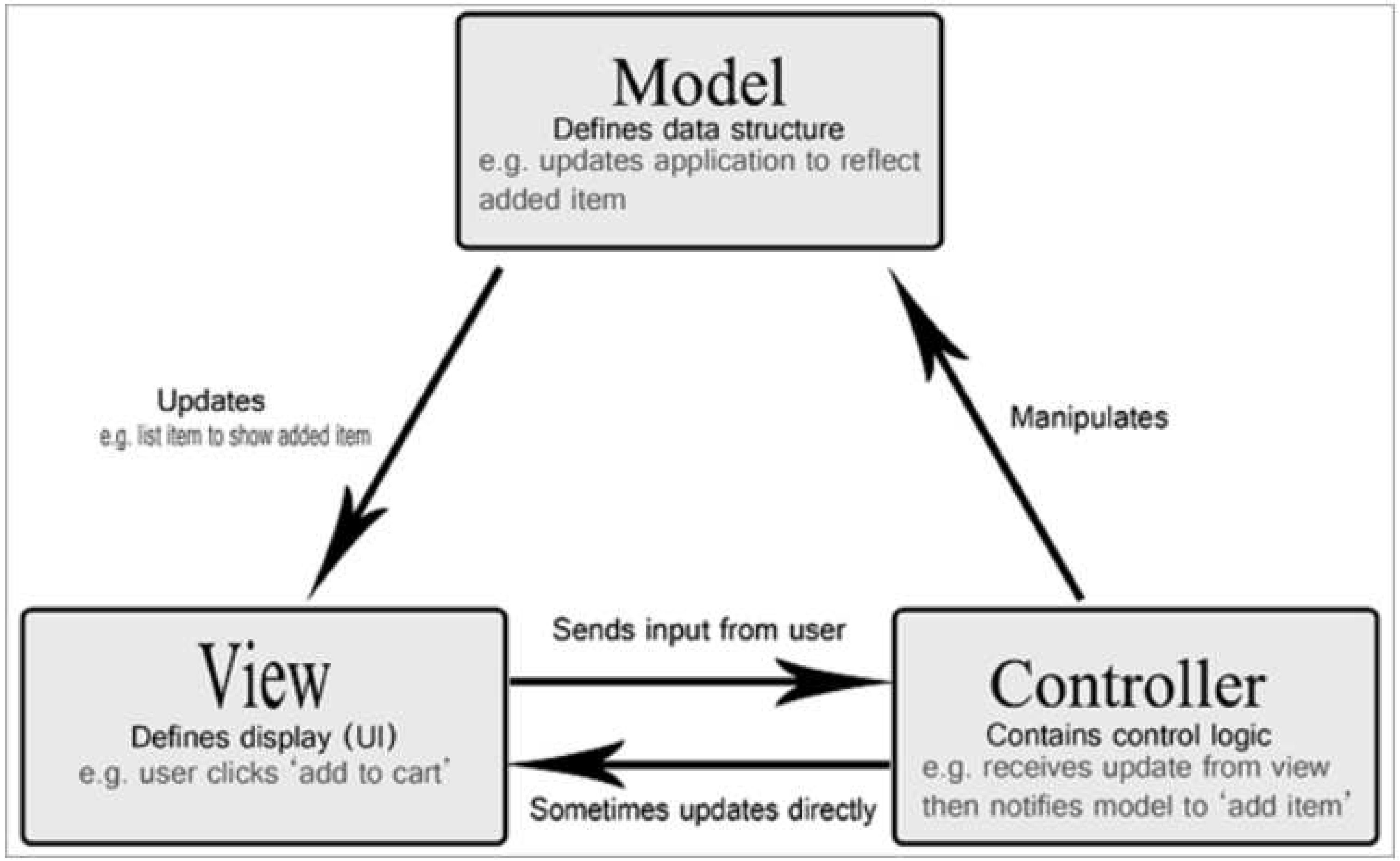
Version	Release date	PHP version
1.0	June 2011	
2.0	September 2011	
3.0	February 22, 2012	
3.1	March 27, 2012	
3.2	May 22, 2012	
4.0	May 28, 2013	≥ 5.3.0
4.1	December 12, 2013	≥ 5.3.0
4.2	June 1, 2014	≥ 5.4.0
5.0	February 4, 2015	≥ 5.4.0
5.1 LTS	June 9, 2015	≥ 5.5.9
5.2	December 21, 2015	≥ 5.5.9
5.3	August 23, 2016	≥ 5.6.4
5.4	January 24, 2017	≥ 5.6.4
5.5 LTS	August 30, 2017	≥ 7.0.0
5.6	February 7, 2018	≥ 7.1.3
5.7	September 4, 2018	≥ 7.1.3
5.8	February 26, 2019	≥ 7.1.3
6 LTS	September 3, 2019	≥ 7.2 and ≤ 8.0[22]
7	March 3, 2020[23]	≥ 7.2 and ≤ 8.0[22]
8	September 8, 2020	≥ 7.3 and ≤ 8.1[22]
9	February 8, 2022[22]	≥ 8.0 and ≤ 8.1[22]

MVC Architecture Of Laravel

MVC Architecture Of Laravel

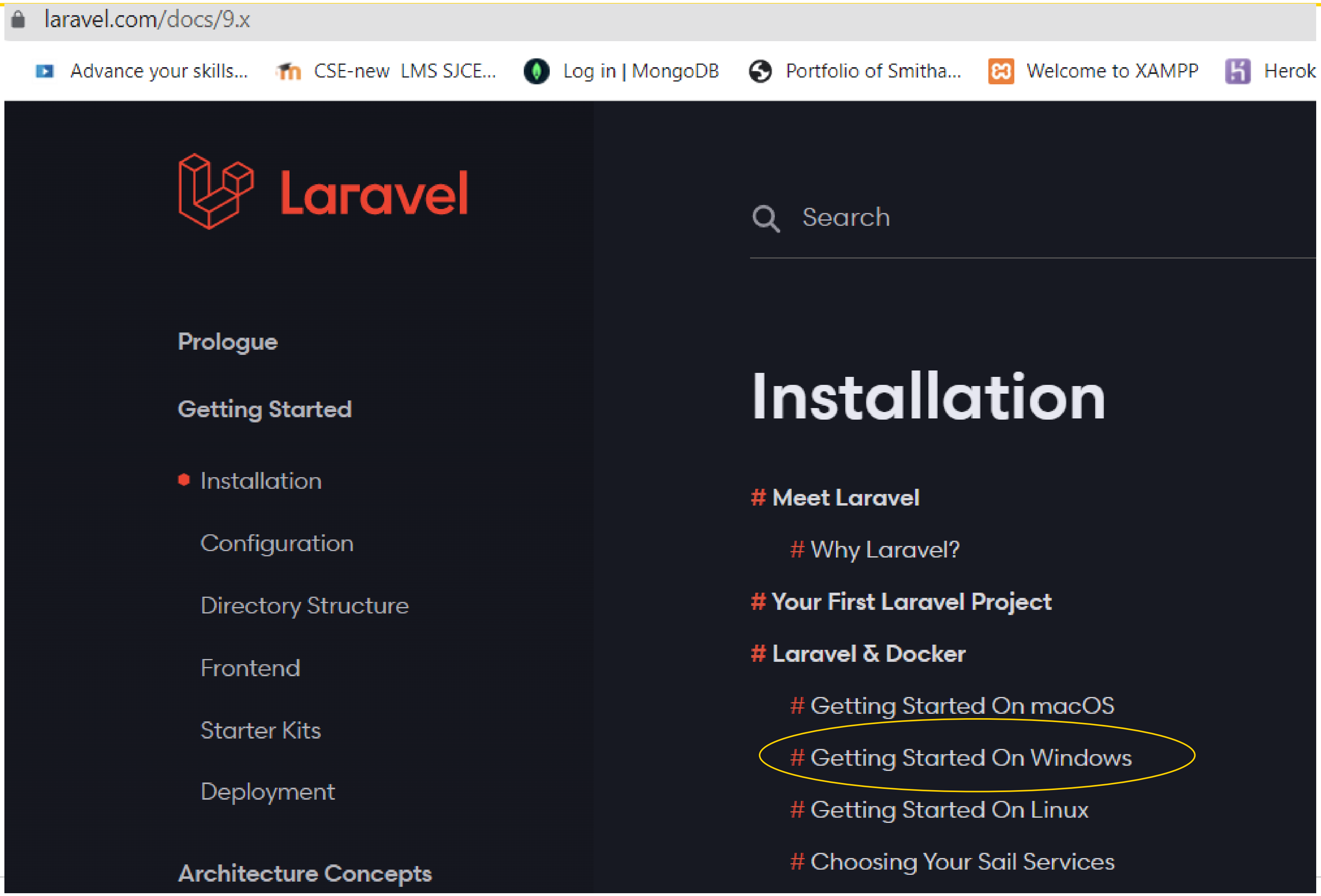
- The Laravel Framework follows **MVC architecture**. MVC is an architectural design pattern that helps to develop web applications faster.
- **MVC** stands for **Model-View-Controller**.
- **Model (M)**–A model handles data used by the web application.
- **View (V)**–A view helps to display data to the user.
- **Controller (C)**–A controller interacts with the model to create data for the view.
- **The following figure shows the interactions between Model, View, and Controller.**

MVC Architecture Of Laravel


Model
Defines data structure
e.g. updates application to reflect added item
Updates
e.g. list item to show added item
Manipulates
View
Defines display (UI)
e.g. user clicks 'add to cart'
Sends input from user
Sometimes updates directly
Controller
Contains control logic
e.g. receives update from view then notifies model to 'add item'

Setting up a Laravel Development Environment

Laravel 9.x


laravel.com/docs/9.x

Advance your skills... CSE-new LMS SJCE... Log in | MongoDB Portfolio of Smitha... Welcome to XAMPP Herok

Laravel

Search

Prologue

Getting Started

● Installation

Configuration

Directory Structure

Frontend

Starter Kits

Deployment

Architecture Concepts

Installation

Meet Laravel

Why Laravel?

Your First Laravel Project

Laravel & Docker

Getting Started On macOS

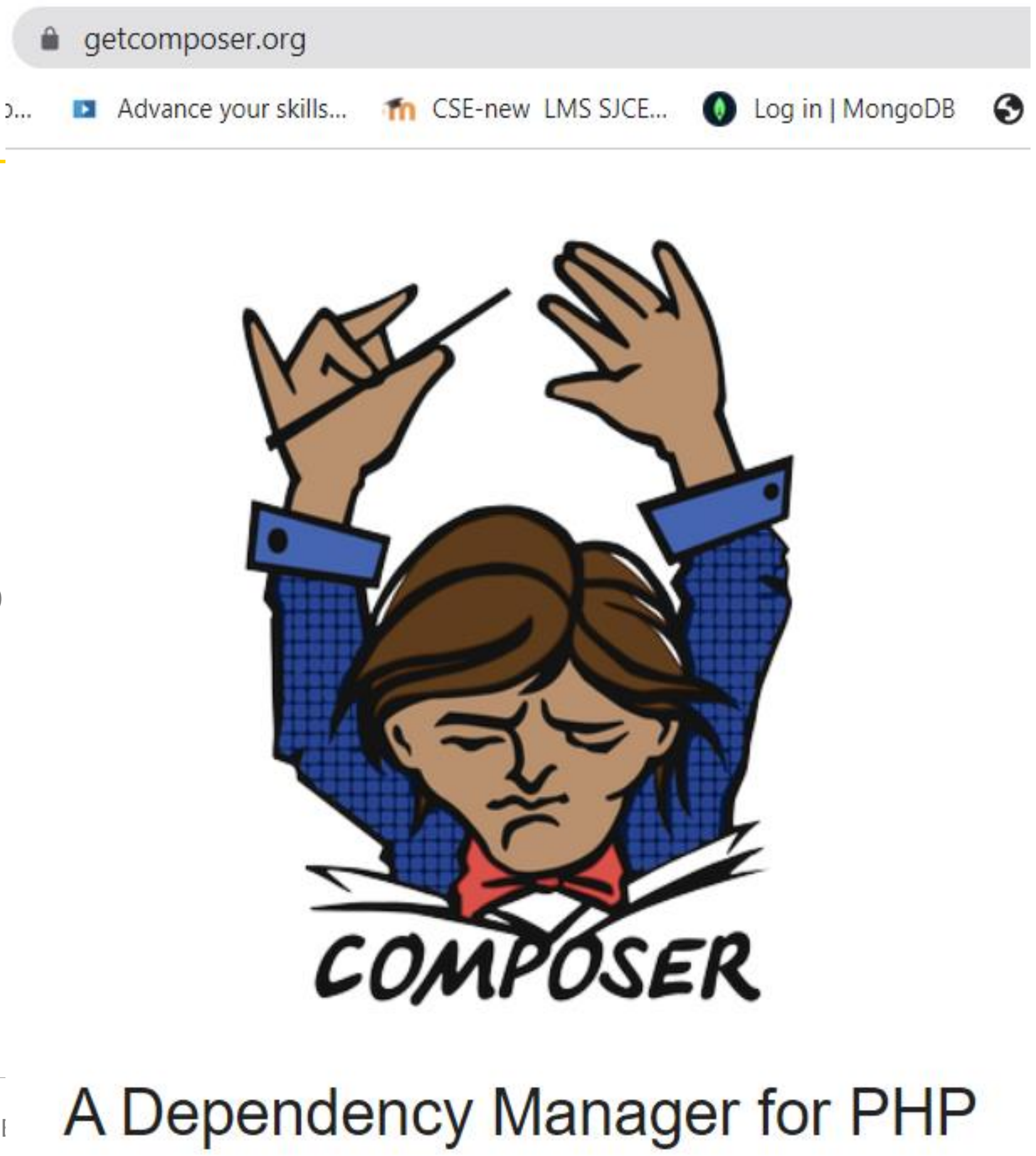
Getting Started On Windows

Getting Started On Linux

Choosing Your Sail Services

composer installation

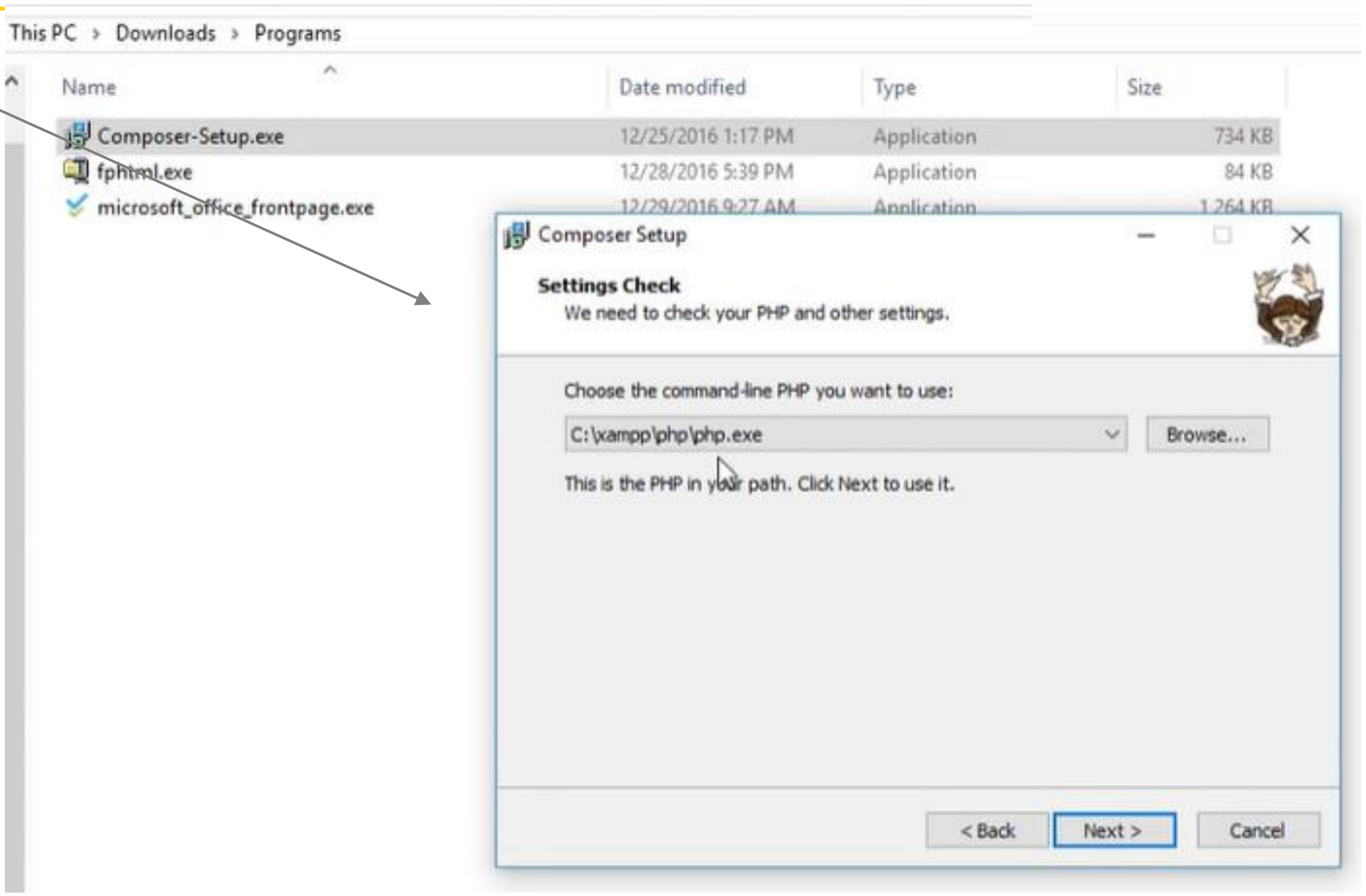
- Before creating your first Laravel project, you should ensure that your local machine has PHP and Composer installed.
- The installer - which requires that you have PHP already installed - will download Composer for you and set up your PATH environment variable so you can simply call composer from any directory.
- Download and run Composer-Setup.exe - it will install the latest composer version whenever it is executed.



A Dependency Manager for PHP

Installation of Composer and laravel

- Composer is a **dependency manager for PHP**. Laravel uses the **Composer** to manage its dependencies.
- You need to install the composer before installing Laravel
- Install Composer-setup.exe
- Composer doesn't know how to install Laravel.so install Laravel installer using the command

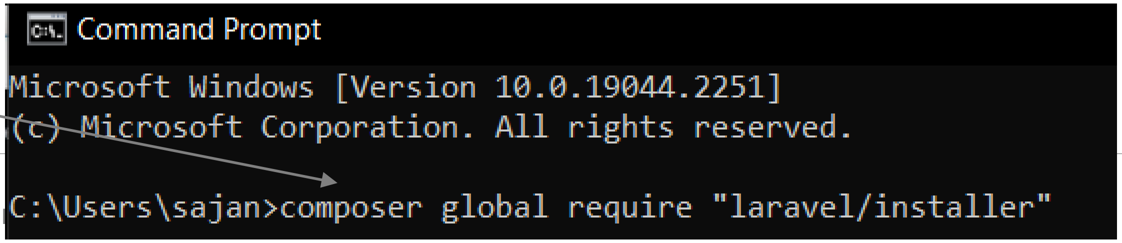

This PC › Downloads › Programs

Name	Date modified	Type	Size
Composer-Setup.exe	12/25/2016 1:17 PM	Application	734 KB
fphtml.exe	12/28/2016 5:39 PM	Application	84 KB
microsoft_office_frontpage.exe	12/29/2016 9:27 AM	Application	1,264 KB

Composer Setup

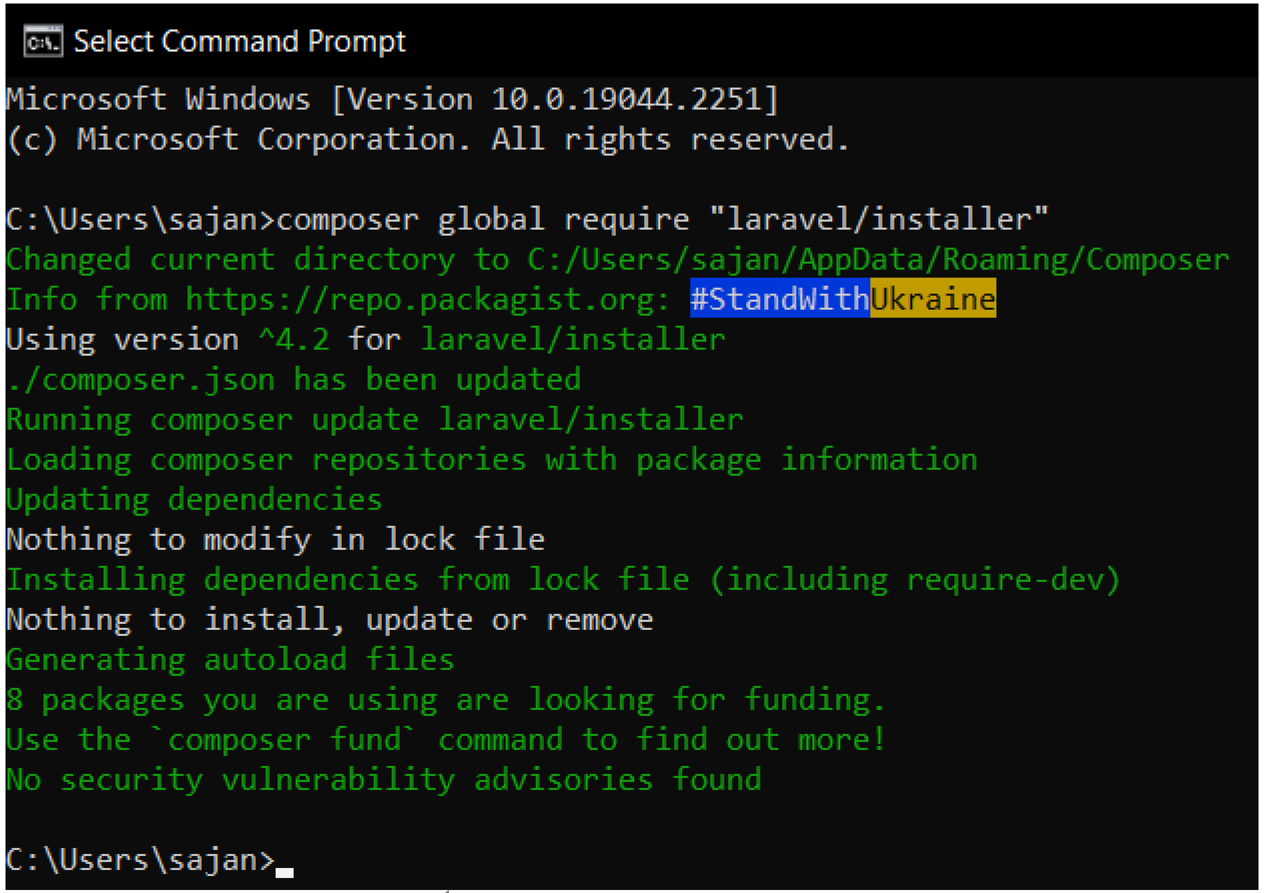
Settings Check
We need to check your PHP and other settings.

Choose the command-line PHP you want to use:
C:\xampp\php\php.exe
This is the PHP in your path. Click Next to use it.

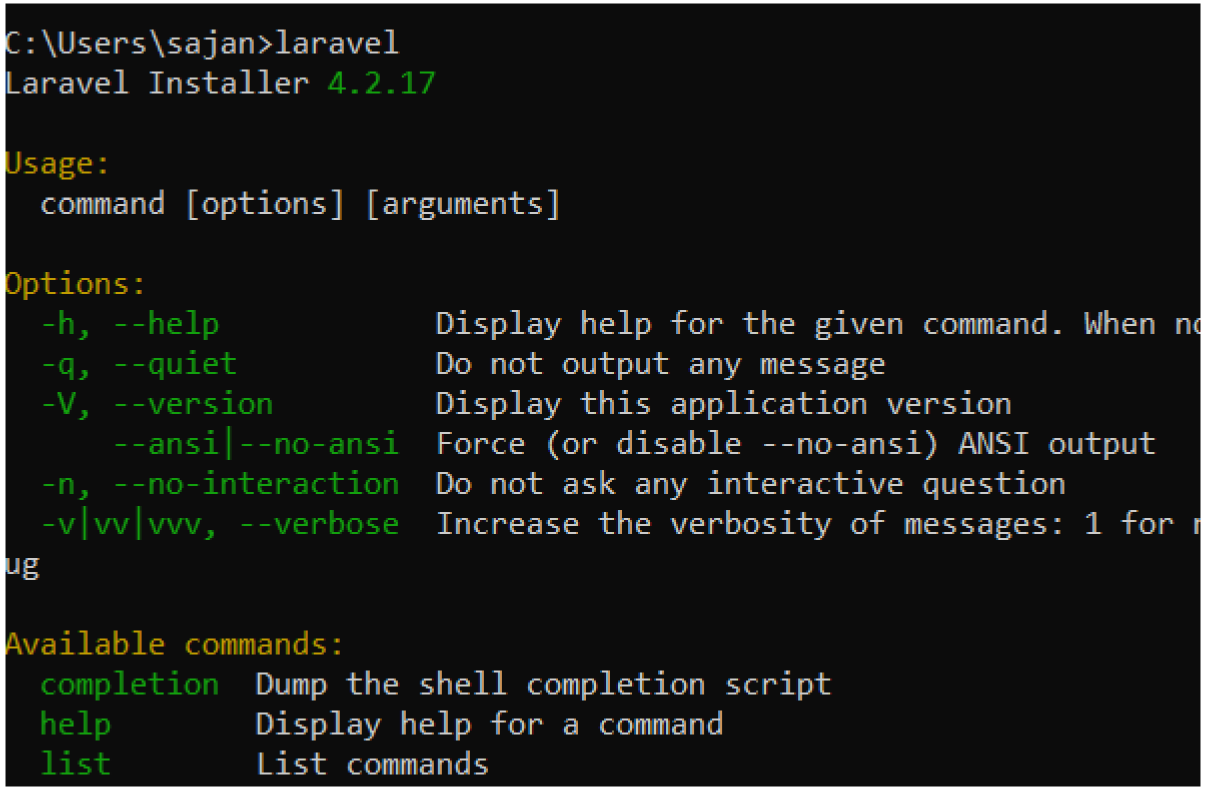

Command Prompt

```
Microsoft Windows [Version 10.0.19044.2251]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sajan>composer global require "laravel/installer"
```

- Check the installation using the command 'laravel'



```
Select Command Prompt
Microsoft Windows [Version 10.0.19044.2251]
(c) Microsoft Corporation. All rights reserved.

C:\Users\sajan>composer global require "laravel/installer"
Changed current directory to C:/Users/sajan/AppData/Roaming/Composer
Info from https://repo.packagist.org: #StandWithUkraine
Using version ^4.2 for laravel/installer
./composer.json has been updated
Running composer update laravel/installer
Loading composer repositories with package information
Updating dependencies
Nothing to modify in lock file
Installing dependencies from lock file (including require-dev)
Nothing to install, update or remove
Generating autoload files
8 packages you are using are looking for funding.
Use the `composer fund` command to find out more!
No security vulnerability advisories found

C:\Users\sajan>_
```



```
C:\Users\sajan>laravel
Laravel Installer 4.2.17

Usage:
  command [options] [arguments]

Options:
  -h, --help            Display help for the given command. When no
  -q, --quiet           Do not output any message
  -V, --version         Display this application version
      --ansi|--no-ansi  Force (or disable --no-ansi) ANSI output
  -n, --no-interaction  Do not ask any interactive question
  -v|vv|vvv, --verbose  Increase the verbosity of messages: 1 for r
ug

Available commands:
  completion  Dump the shell completion script
  help        Display help for a command
  list        List commands
```

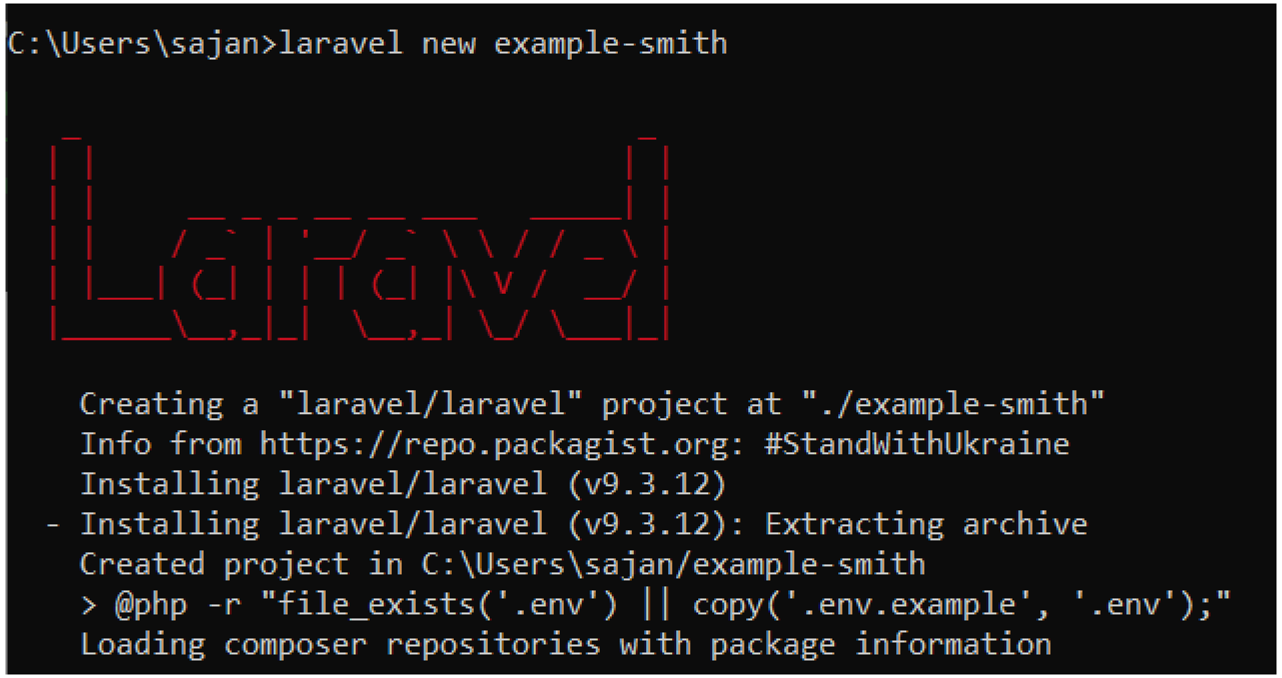
- **Laravel installation is completed**

Create New project –go to command prompt
laravel new example-smith

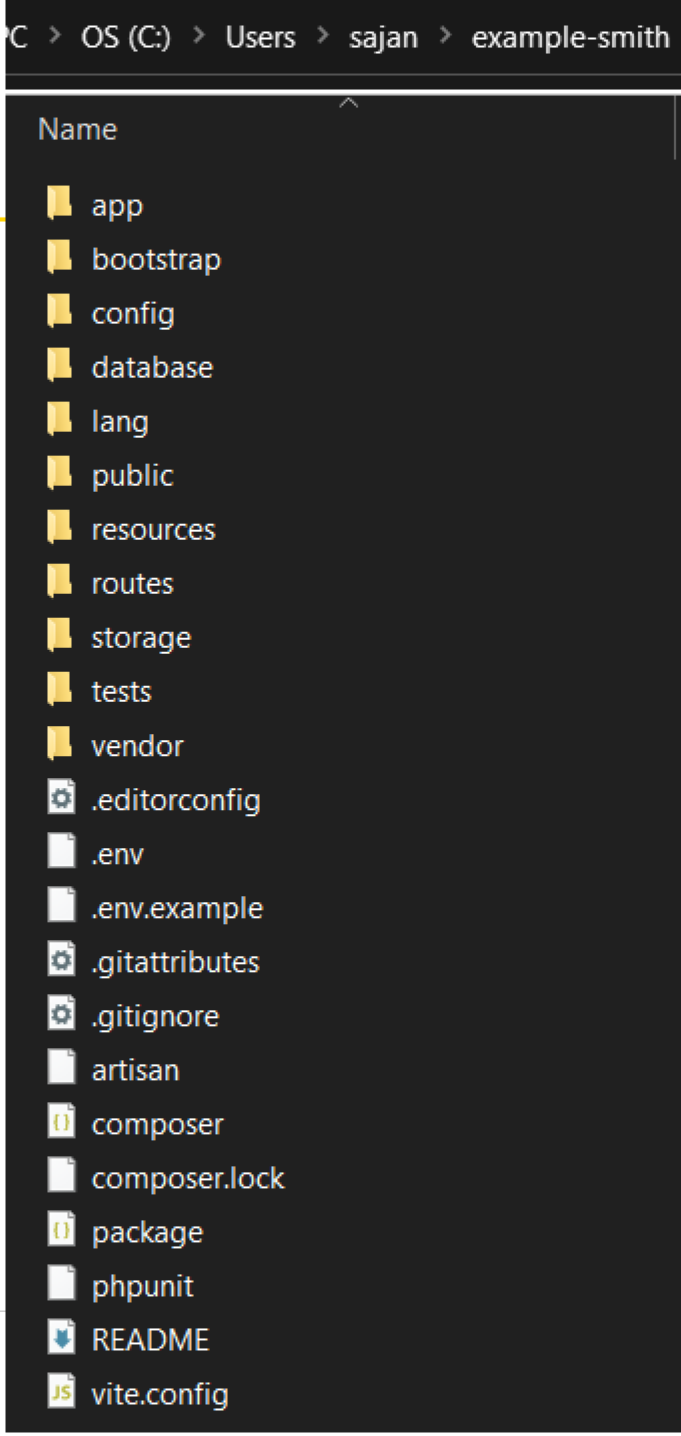
Installation of Composer and laravel

- Open command prompt and type

  `laravel new example-smith`



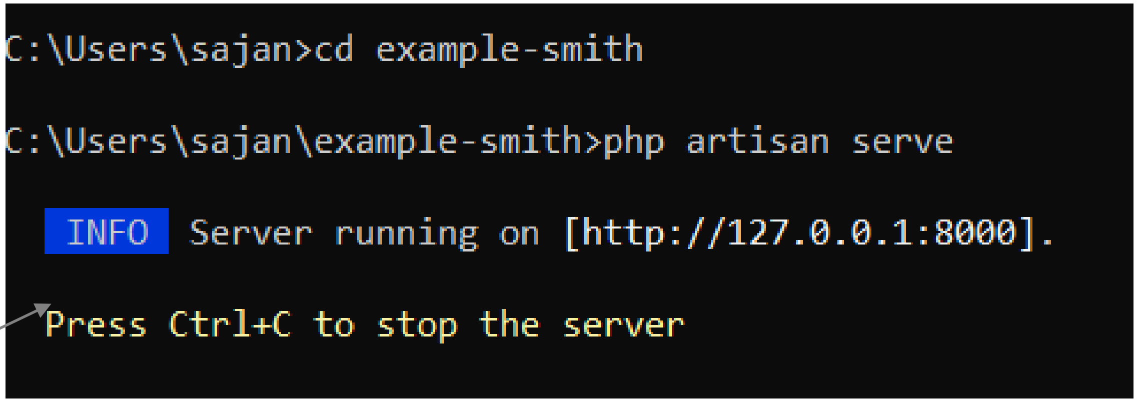
```
C:\Users\sajan>laravel new example-smith

  Creating a "laravel/laravel" project at "./example-smith"
  Info from https://repo.packagist.org: #StandWithUkraine
  Installing laravel/laravel (v9.3.12)
- Installing laravel/laravel (v9.3.12): Extracting archive
  Created project in C:\Users\sajan/example-smith
  > @php -r "file_exists('.env') || copy('.env.example', '.env');"
  Loading composer repositories with package information
```



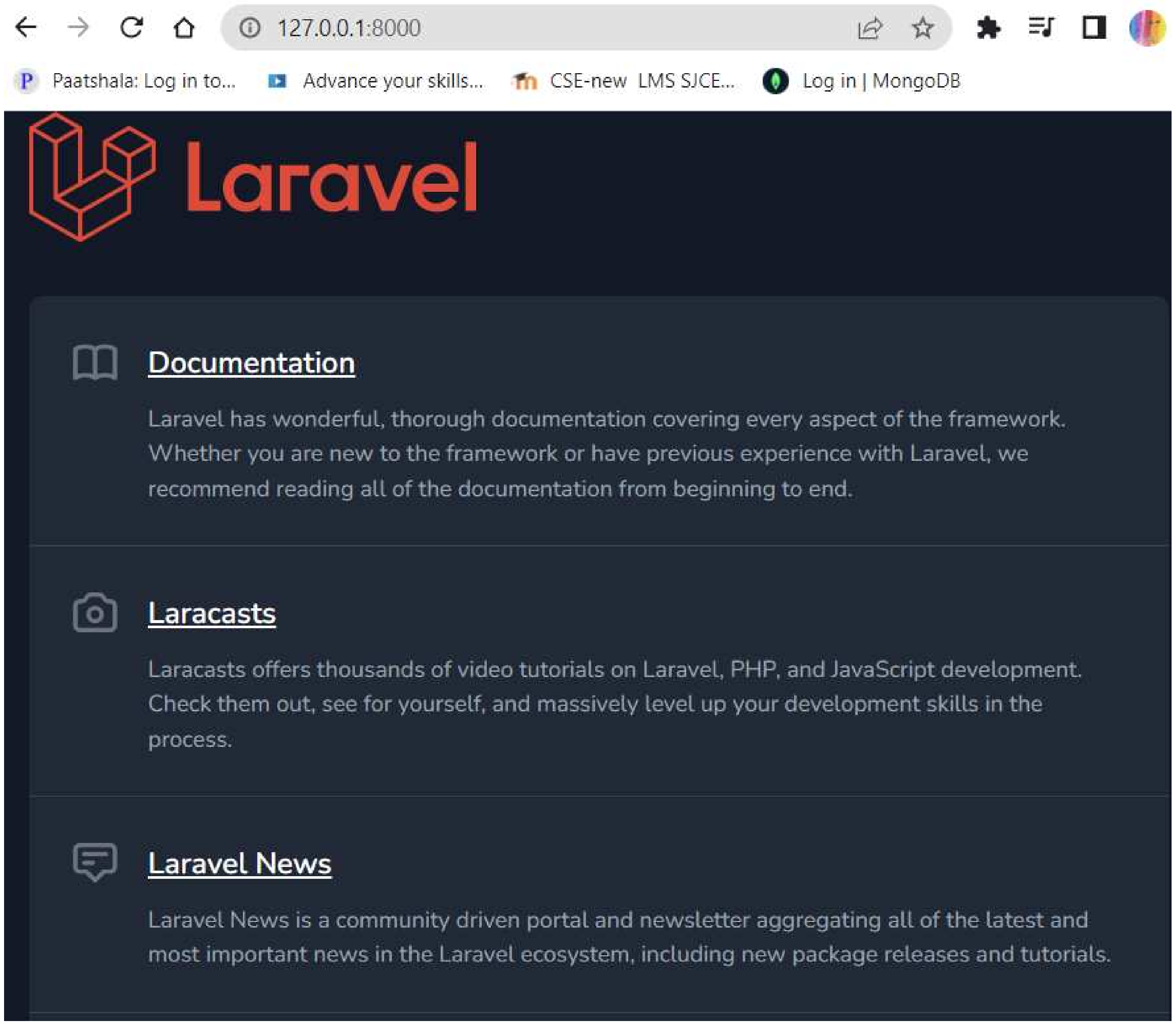
```
C > OS (C:) > Users > sajan > example-smith

Name
app
bootstrap
config
database
lang
public
resources
routes
storage
tests
vendor
.editorconfig
.env
.env.example
.gitattributes
.gitignore
artisan
composer
composer.lock
package
phpunit
README
vite.config
```

installation of Laravel

- After the project has been created, start Laravel's local development server using the Laravel's Artisan CLI serve command:
- Once you have started the Artisan development server, your application will be accessible in your web browser at [http://127.0.0.1:8000]
- Laravel web server is ready at this URL.
- As Laravel supports MVC ,it is the “web.php” file of routes directory ,displays the welcome page contents from “ views/welcome.blade.php”



```
C:\Users\sajan>cd example-smith

C:\Users\sajan\example-smith>php artisan serve

  INFO  Server running on [http://127.0.0.1:8000].

  Press Ctrl+C to stop the server
```



Documentation

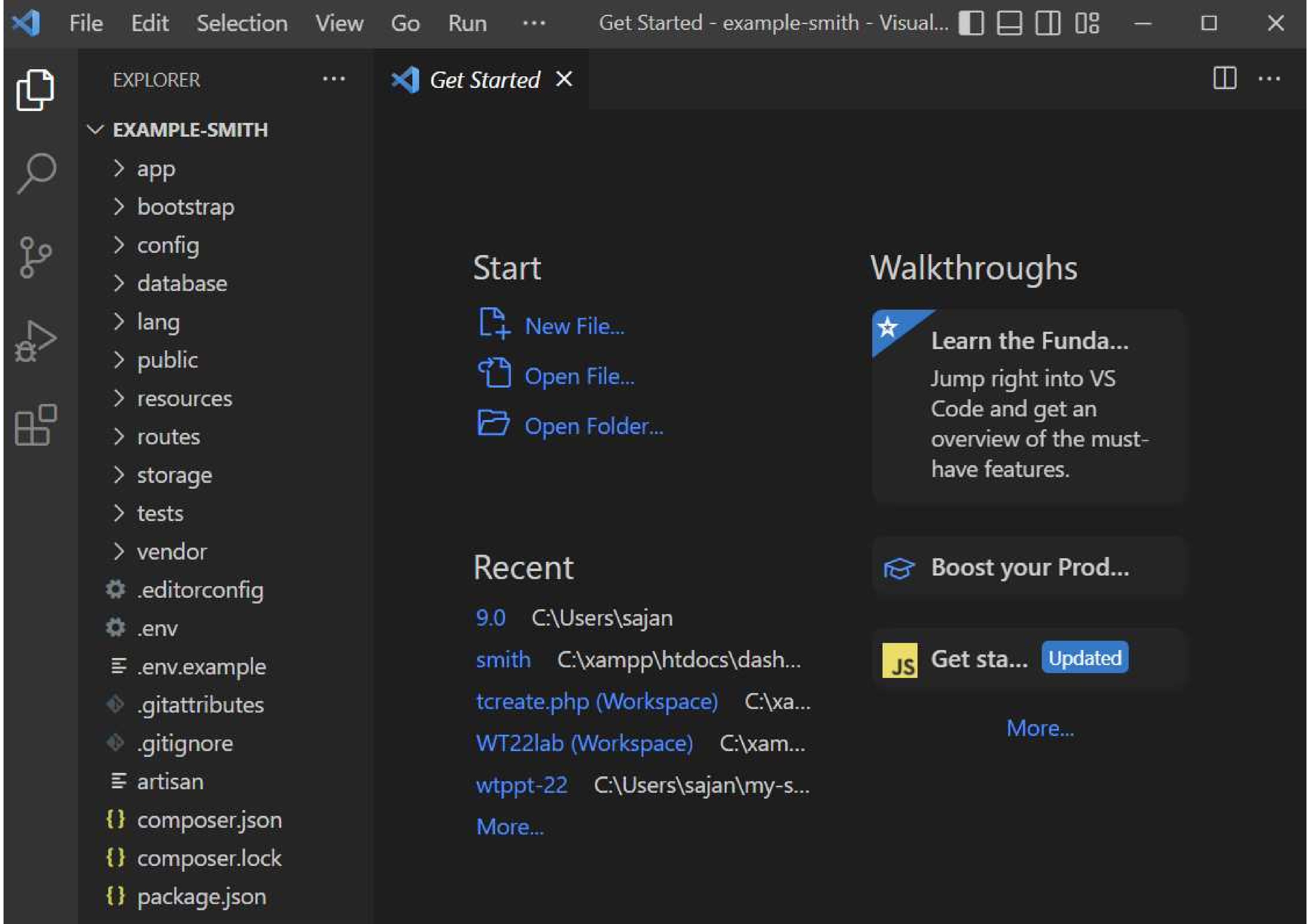
Laravel has wonderful, thorough documentation covering every aspect of the framework. Whether you are new to the framework or have previous experience with Laravel, we recommend reading all of the documentation from beginning to end.

Laracasts

Laracasts offers thousands of video tutorials on Laravel, PHP, and JavaScript development. Check them out, see for yourself, and massively level up your development skills in the process.

Laravel News

Laravel News is a community driven portal and newsletter aggregating all of the latest and most important news in the Laravel ecosystem, including new package releases and tutorials.



EXAMPLE-SMITH
- app
- bootstrap
- config
- database
- lang
- public
- resources
- routes
- storage
- tests
- vendor
- .editorconfig
- .env
- .env.example
- .gitattributes
- .gitignore
- artisan
- composer.json
- composer.lock
- package.json

Start

New File...

Open File...

Open Folder...

Recent

9.0 C:\Users\sajan

smith C:\xampp\htdocs\dash...

tcreate.php (Workspace) C:\xa...

WT22lab (Workspace) C:\xam...

wtppt-22 C:\Users\sajan\my-s...

More...

Walkthroughs

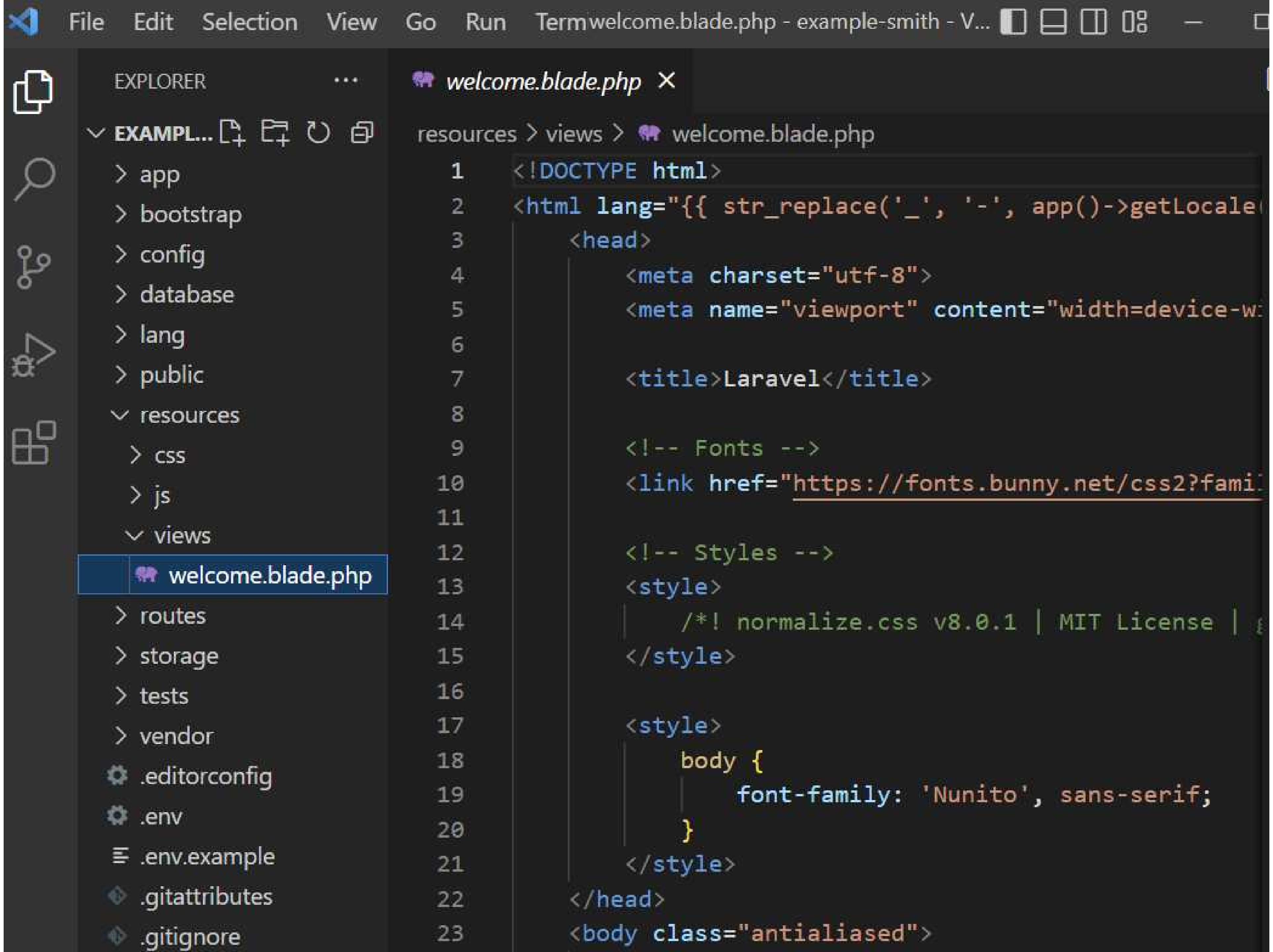
Learn the Funda...

Jump right into VS Code and get an overview of the must-have features.

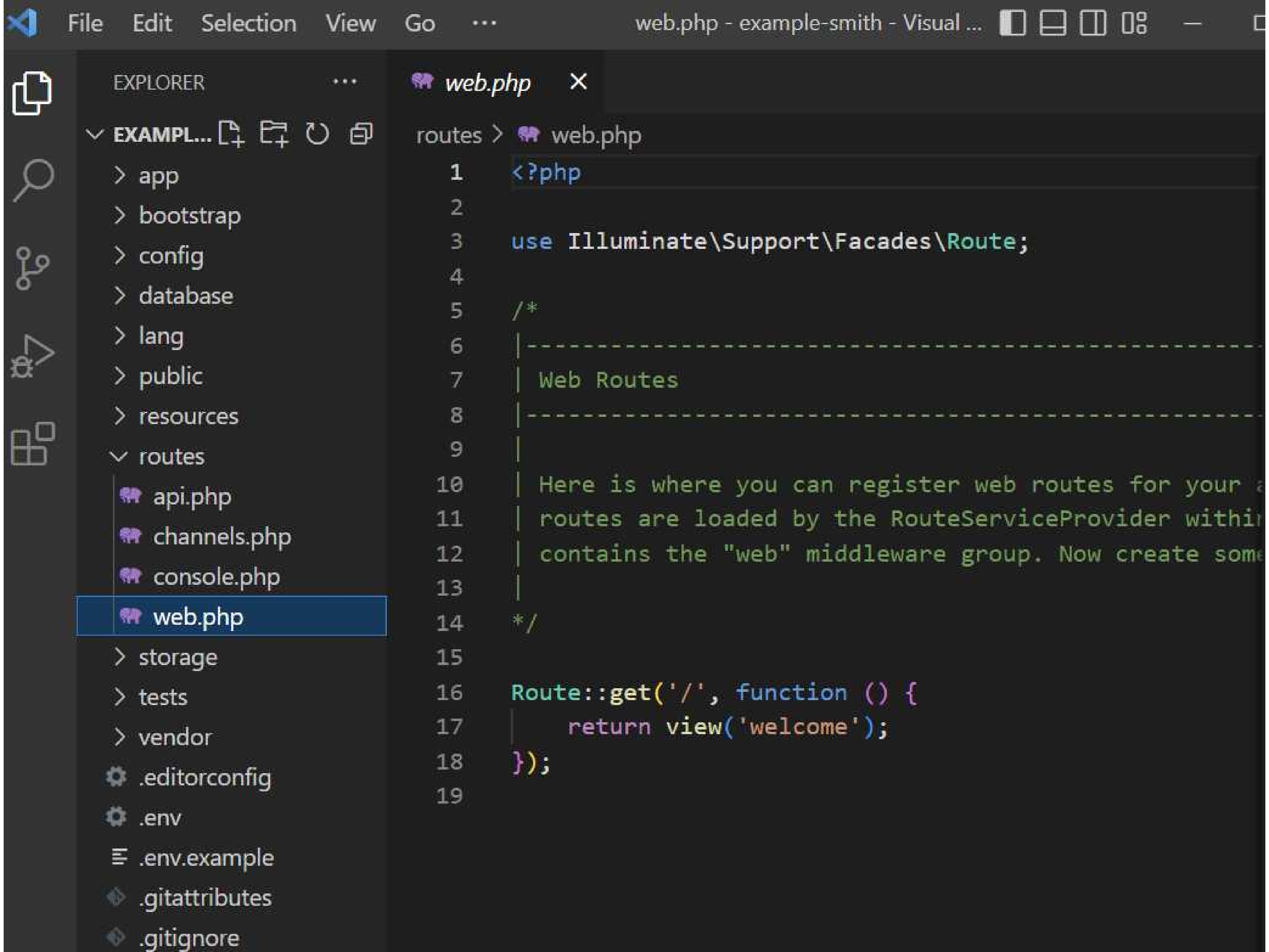
Boost your Prod...

Get sta... Updated

More...



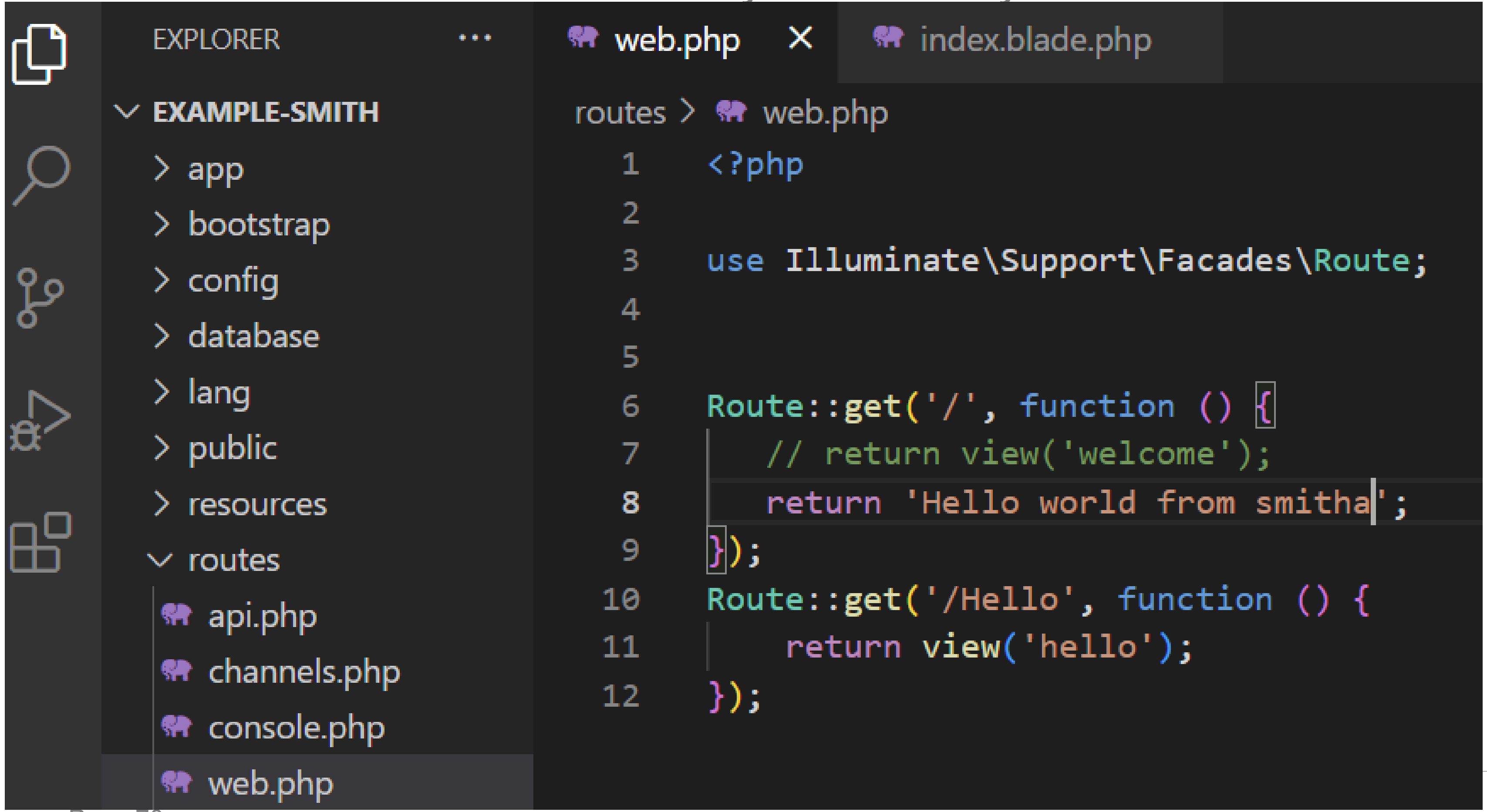
```
<!DOCTYPE html>
<html lang="{{ str_replace('_', '-', app()->getLocale(
    <head>
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-wi

        <title>Laravel</title>

        <!-- Fonts -->
        <link href="https://fonts.bunny.net/css2?famil

        <!-- Styles -->
        <style>
            /*! normalize.css v8.0.1 | MIT License | g
        </style>

        <style>
            body {
                font-family: 'Nunito', sans-serif;
            }
        </style>
    </head>
    <body class="antialiased">
```



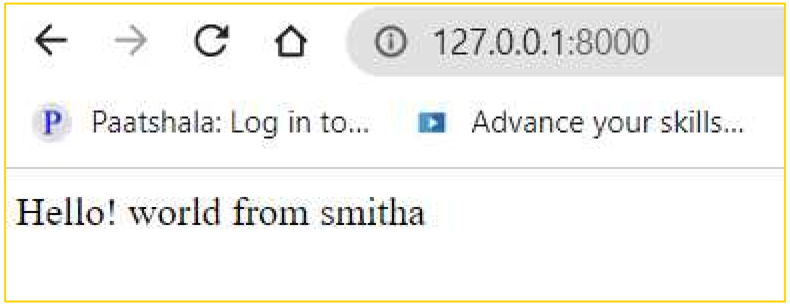
```php
<?php

use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your a
| routes are loaded by the RouteServiceProvider withi
| contains the "web" middleware group. Now create some
|
*/

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

```

Route Definitions

- In a Laravel application, you will define your “web” routes in routes/web.php and your “API” routes in routes/api.php. Web routes are those that will be visited by your end users; API routes are those for your API, if you have one



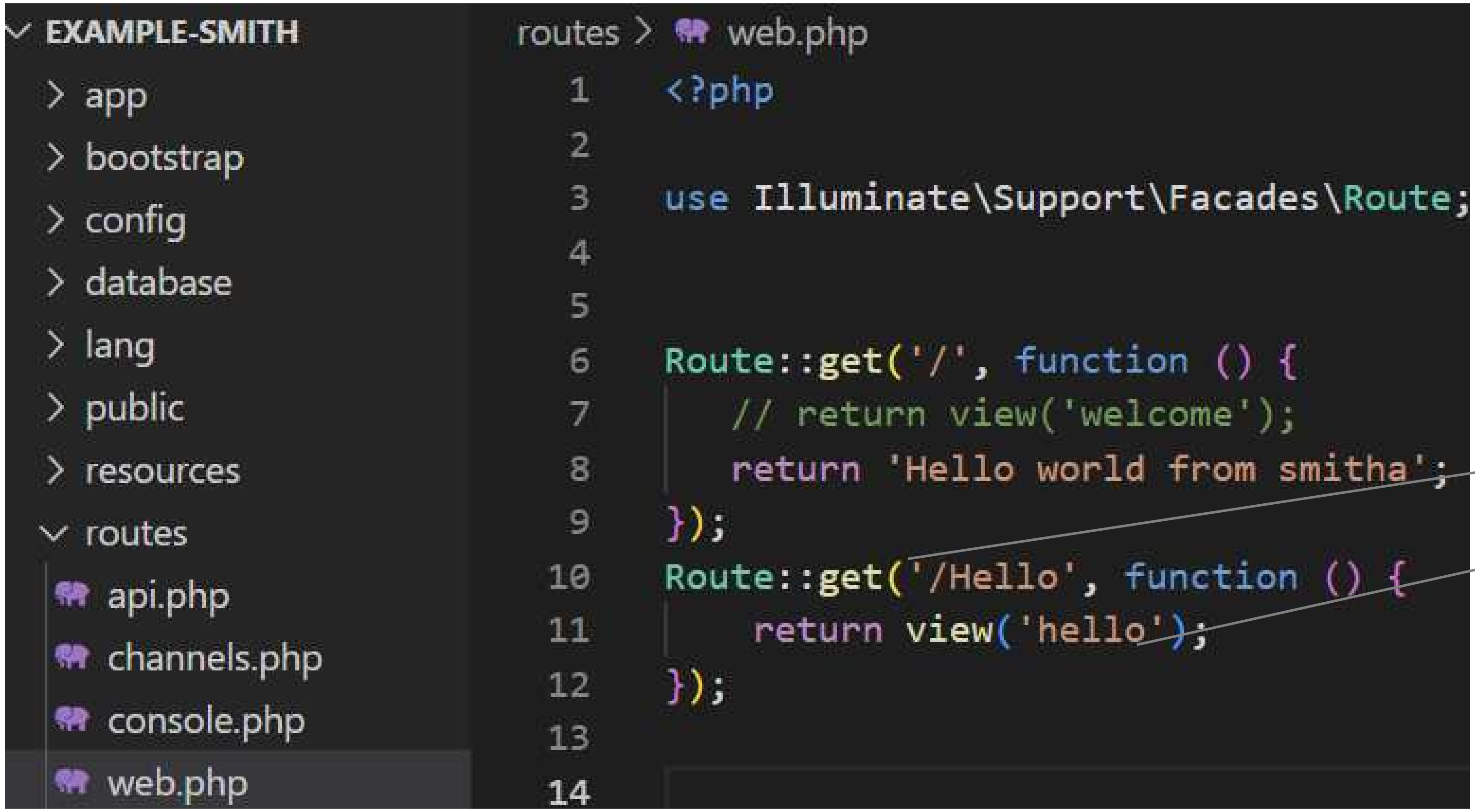
```php
<?php

use Illuminate\Support\Facades\Route;


Route::get('/', function () {
    // return view('welcome');
    return 'Hello world from smitha';
});
Route::get('/Hello', function () {
     return view('hello');
});
```



```
Hello! world from smitha
```

Route Definitions-using view

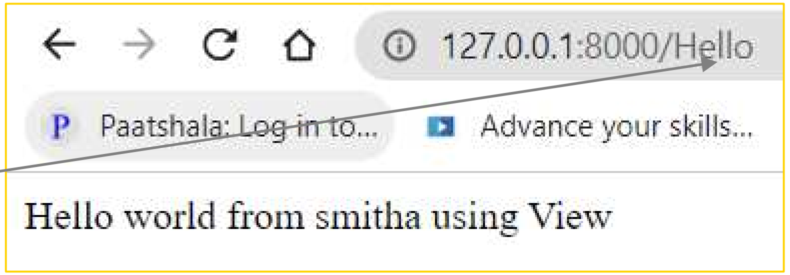
- In a Laravel application, you will define your "web" routes in routes/web.php



```
EXAMPLE-SMITH
> app
> bootstrap
> config
> database
> lang
> public
> resources
v routes
    api.php
    channels.php
    console.php
    web.php
```

routes > web.php

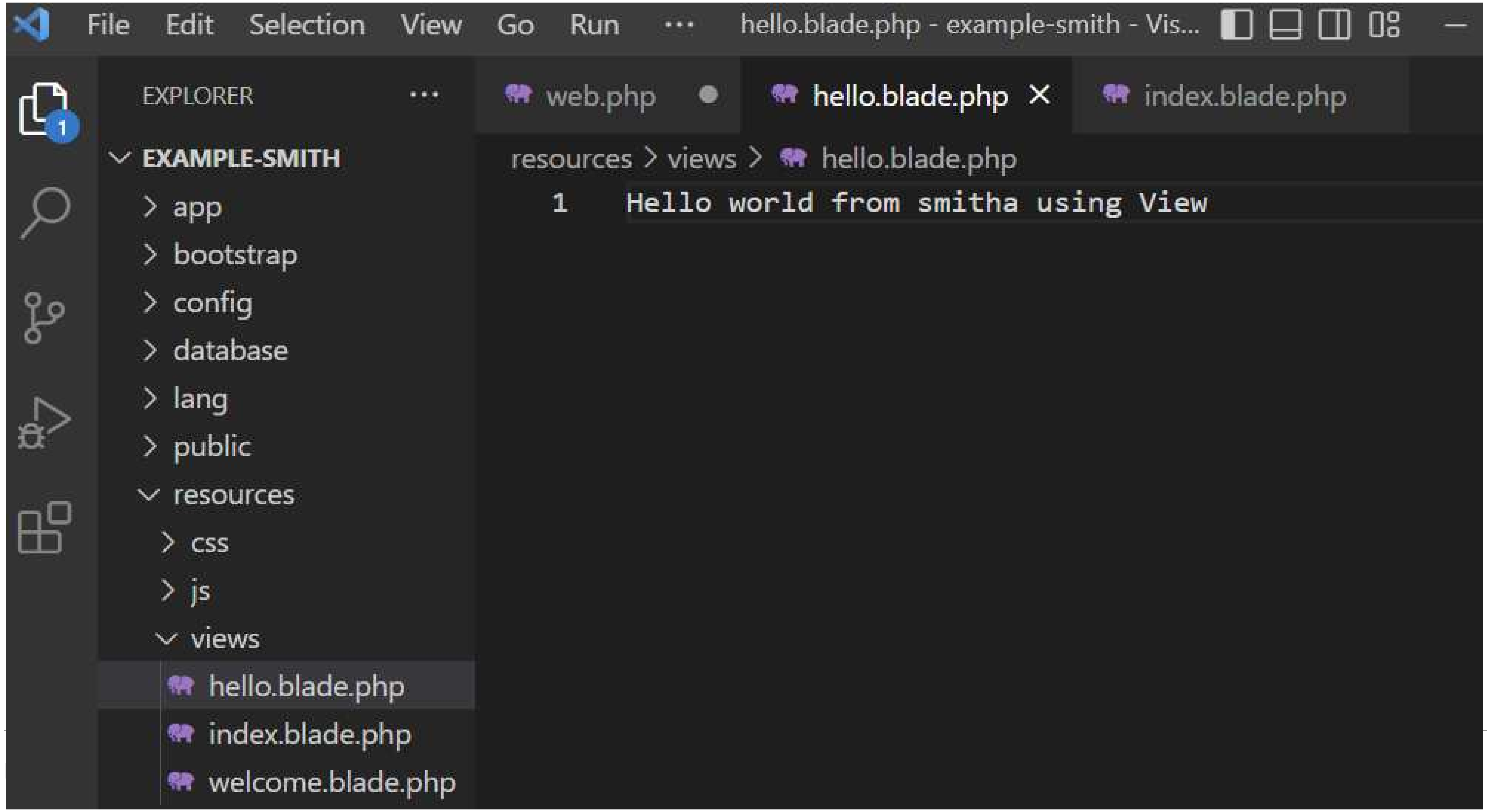
```php
<?php

use Illuminate\Support\Facades\Route;


Route::get('/', function () {
    // return view('welcome');
    return 'Hello world from smitha';
});
Route::get('/Hello', function () {
     return view('hello');
});
```



```
127.0.0.1:8000/Hello
Paatshala: Log in to...    Advance your skills...

Hello world from smitha using View
```

Route Definitions-using view

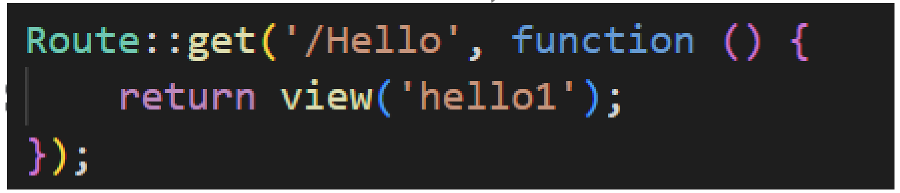
- Define your “webpage ” contents in views/hello.blade.php



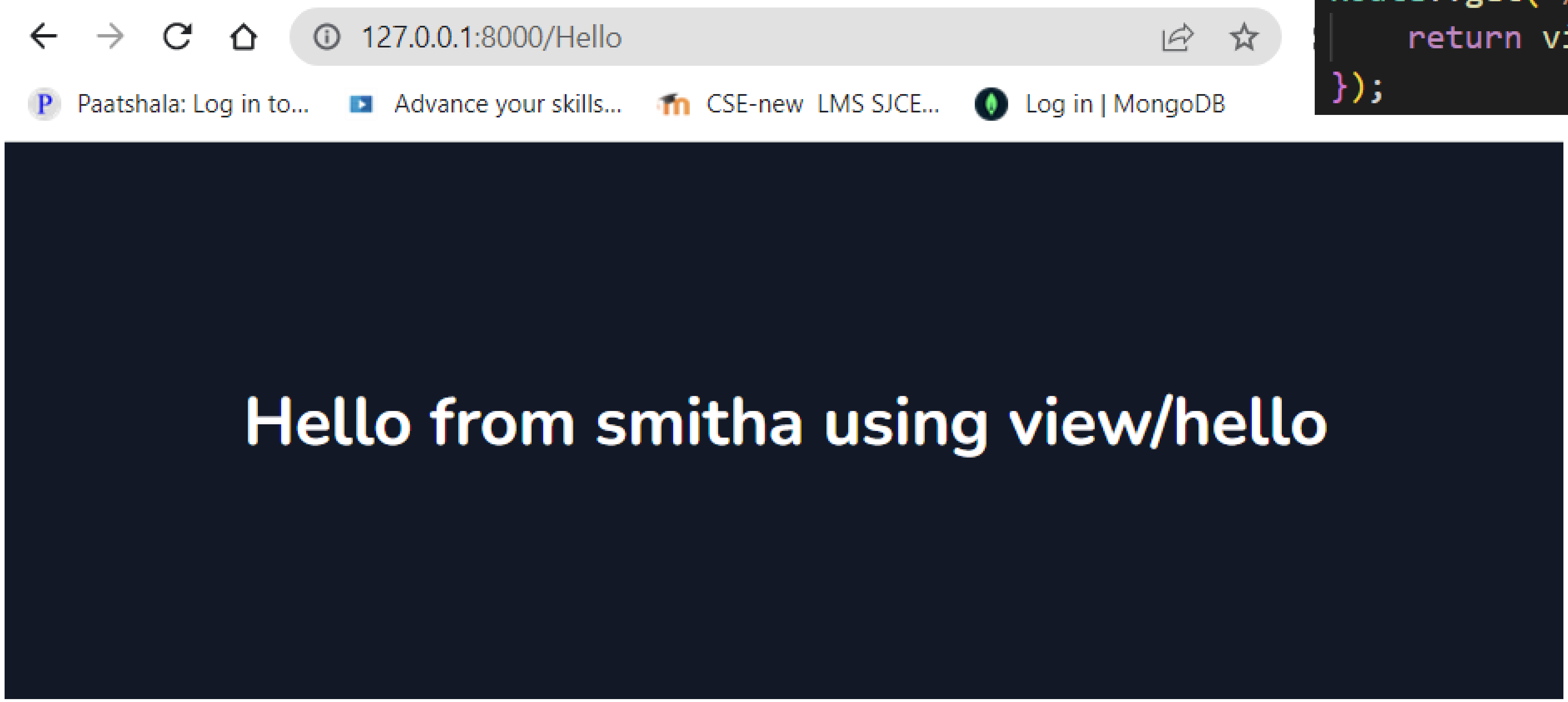
```
Hello world from smitha using View
```

Route Definitions-using view

- Define your "webpage " contents in views/hello1.blade.php
- **web.php**



```
Route::get('/Hello', function () {
    return view('hello1');
});
```



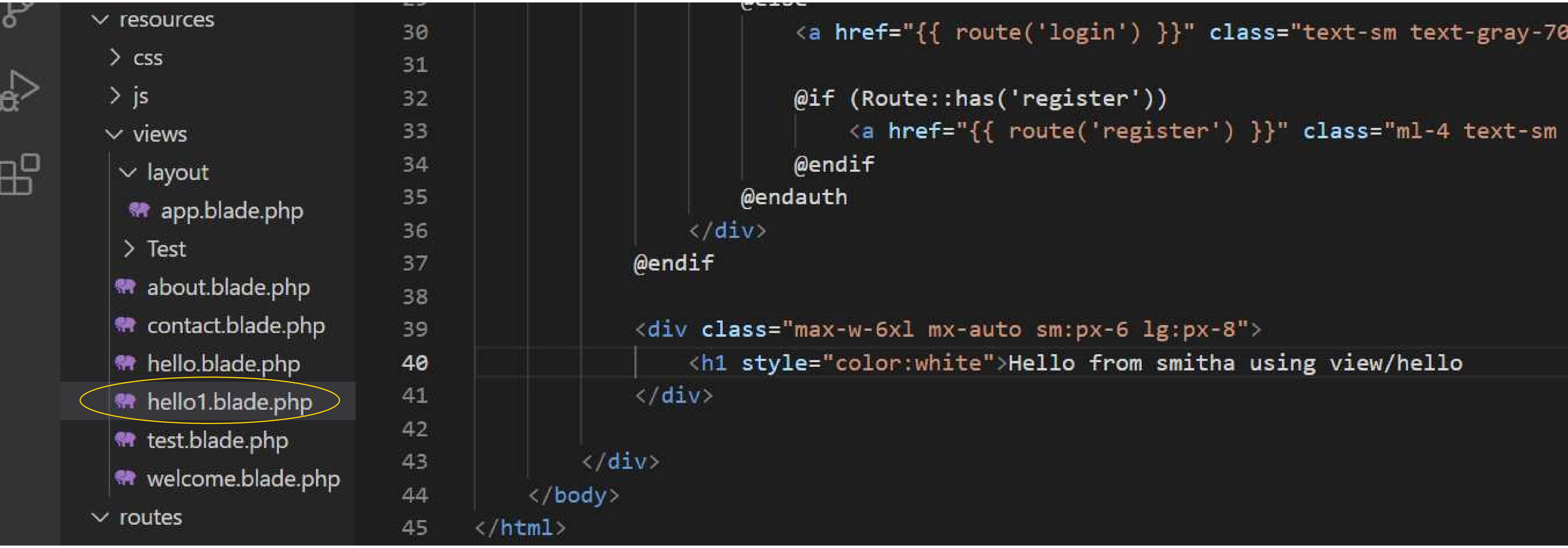
127.0.0.1:8000/Hello

Paatshala: Log in to... Advance your skills... CSE-new LMS SJCE... Log in | MongoDB

Hello from smitha using view/hello

Route Definitions-using view

- Define your “webpage ” contents in views/hello1.blade.php

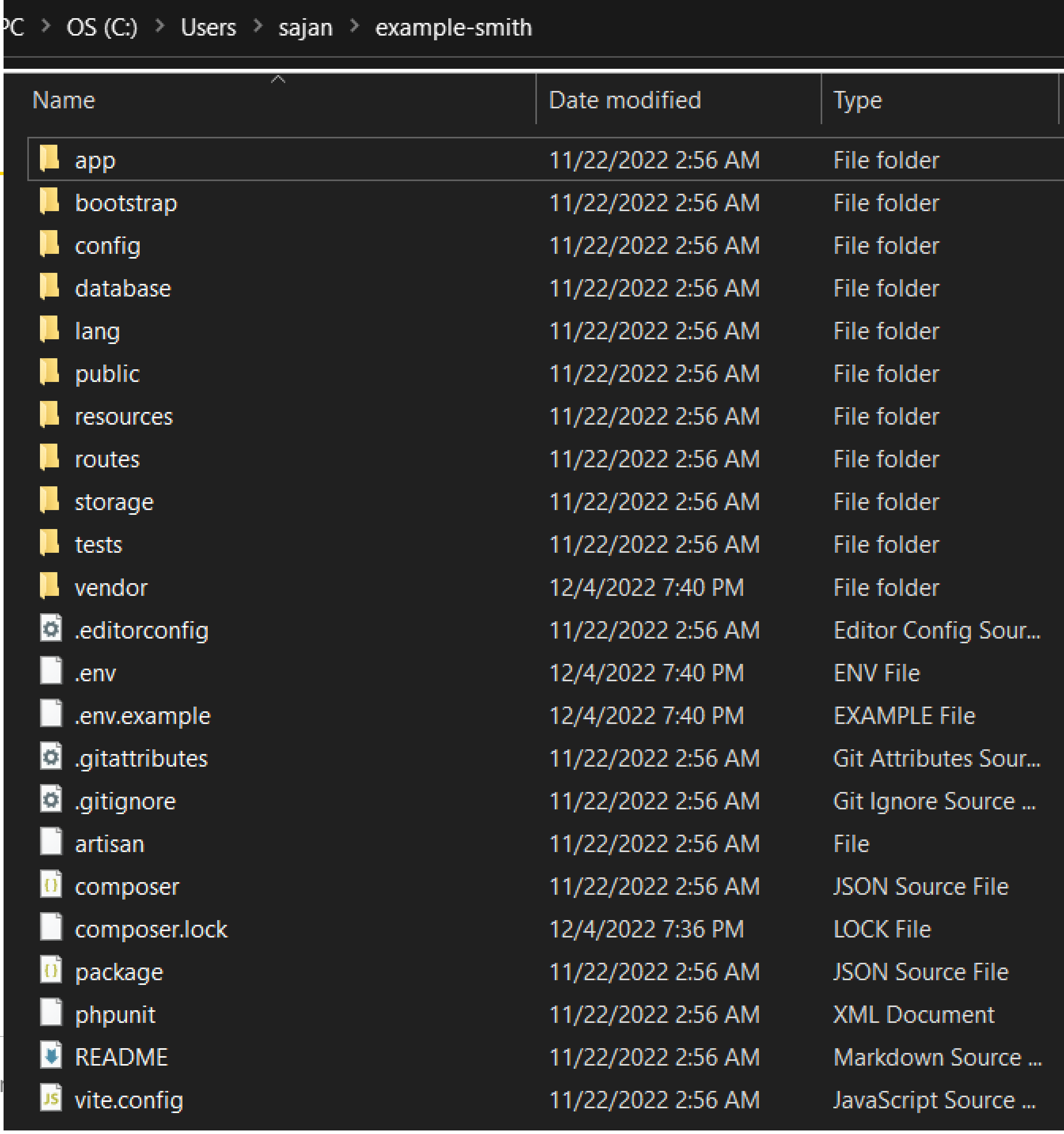


```
                        <a href="{{ route('login') }}" class="text-sm text-gray-70

                        @if (Route::has('register'))
                            <a href="{{ route('register') }}" class="ml-4 text-sm
                        @endif
                    @endauth
                </div>
            @endif

            <div class="max-w-6xl mx-auto sm:px-6 lg:px-8">
                <h1 style="color:white">Hello from smitha using view/hello
            </div>

        </div>
    </body>
</html>
```

Application structure of Laravel

Laravel PHP Project Structure

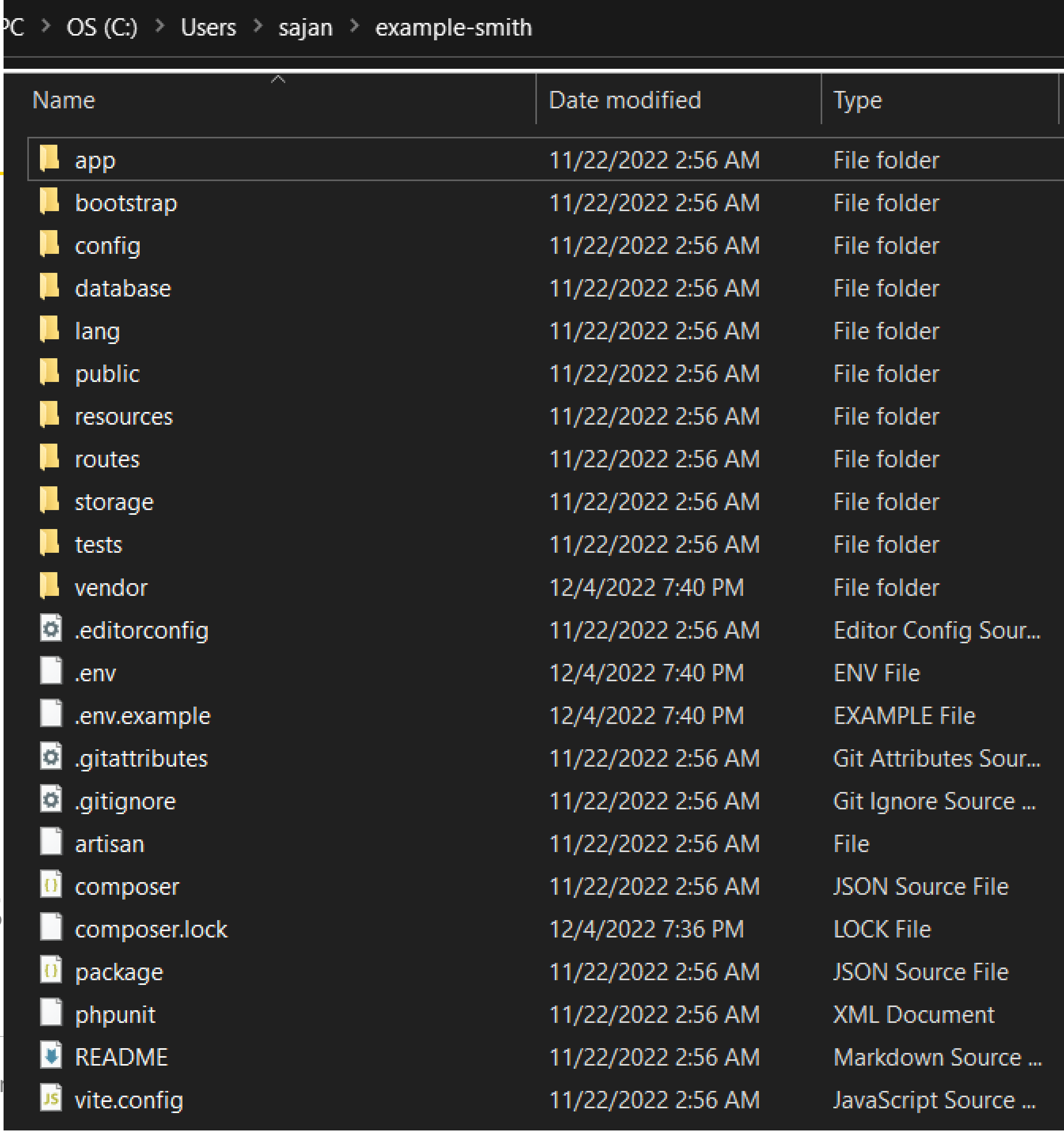
- Open your project from the **File Explorer** to see the root directory structure of the Laravel project.
- **app/Http directory:** This directory contains sub-directories for **Controllers & Middleware.**
- **app/User.php file:** This is the default **Model** provided by Laravel.
- resources/**views directory:** This directory contains **HTML files or templates** which help to display data to the user.
- **routes directory:** This directory contains all the **routes definitions** for the Laravel PHP application.
- The bootstrap directory contains the app.php file which bootstraps the framework.



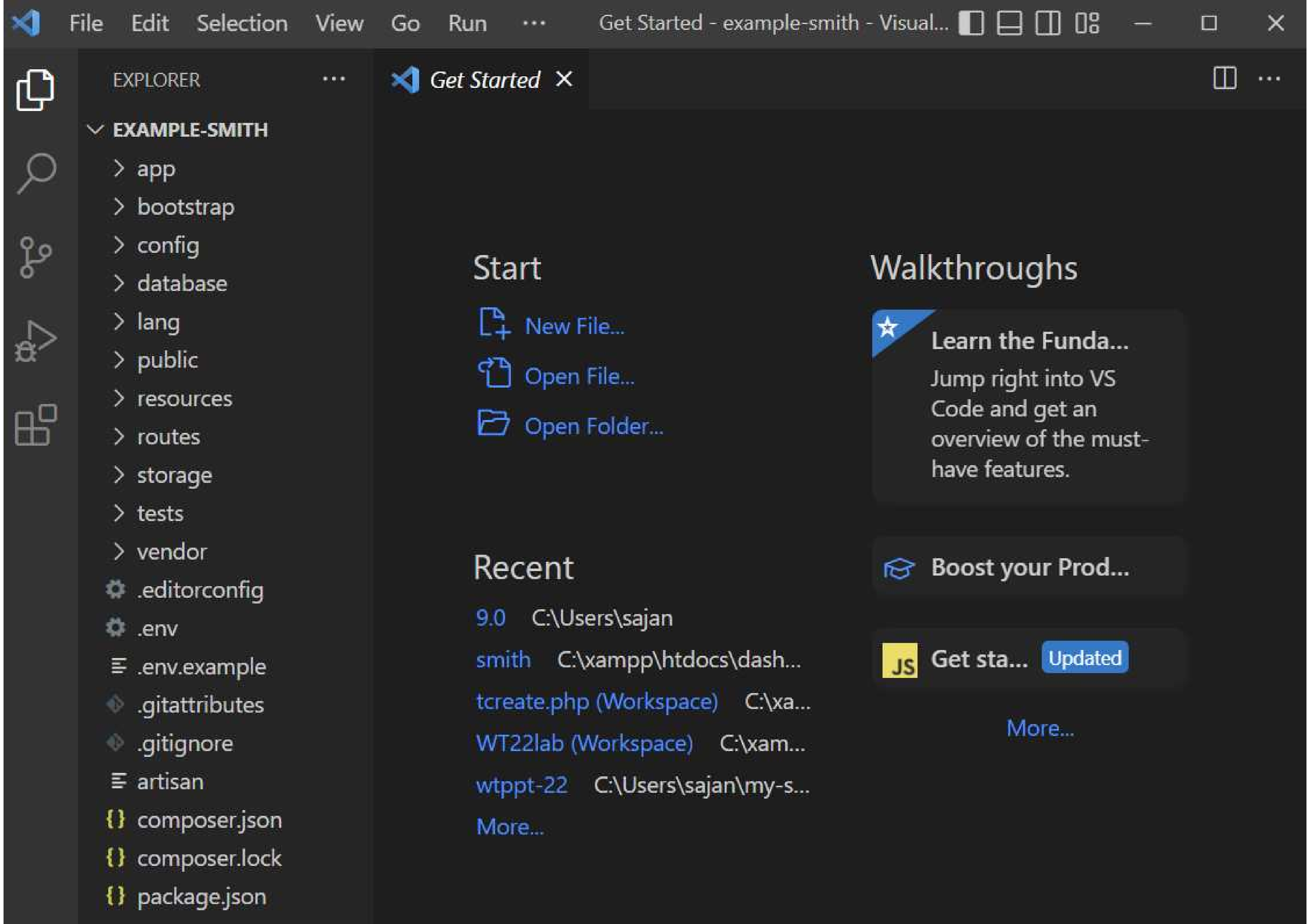
Name	Date modified	Type
app	11/22/2022 2:56 AM	File folder
bootstrap	11/22/2022 2:56 AM	File folder
config	11/22/2022 2:56 AM	File folder
database	11/22/2022 2:56 AM	File folder
lang	11/22/2022 2:56 AM	File folder
public	11/22/2022 2:56 AM	File folder
resources	11/22/2022 2:56 AM	File folder
routes	11/22/2022 2:56 AM	File folder
storage	11/22/2022 2:56 AM	File folder
tests	11/22/2022 2:56 AM	File folder
vendor	12/4/2022 7:40 PM	File folder
.editorconfig	11/22/2022 2:56 AM	Editor Config Sour...
.env	12/4/2022 7:40 PM	ENV File
.env.example	12/4/2022 7:40 PM	EXAMPLE File
.gitattributes	11/22/2022 2:56 AM	Git Attributes Sour...
.gitignore	11/22/2022 2:56 AM	Git Ignore Source ...
artisan	11/22/2022 2:56 AM	File
composer	11/22/2022 2:56 AM	JSON Source File
composer.lock	12/4/2022 7:36 PM	LOCK File
package	11/22/2022 2:56 AM	JSON Source File
phpunit	11/22/2022 2:56 AM	XML Document
README	11/22/2022 2:56 AM	Markdown Source ...
vite.config	11/22/2022 2:56 AM	JavaScript Source ...

Laravel PHP Project Structure

- The config directory, as the name implies, contains all of your application's configuration files.
- The database directory contains your database migrations, model factories, and seeds.
- The lang directory houses all of your application's language files.
- The public directory contains the index.php file, which is the entry point for all requests entering your application and configures autoloading. This directory also houses your assets such as images, JavaScript, and CSS.
- The resources directory contains your views as well as your raw, un-compiled assets such as CSS or JavaScript.


PC › OS (C:) › Users › sajan › example-smith

Name	Date modified	Type
app	11/22/2022 2:56 AM	File folder
bootstrap	11/22/2022 2:56 AM	File folder
config	11/22/2022 2:56 AM	File folder
database	11/22/2022 2:56 AM	File folder
lang	11/22/2022 2:56 AM	File folder
public	11/22/2022 2:56 AM	File folder
resources	11/22/2022 2:56 AM	File folder
routes	11/22/2022 2:56 AM	File folder
storage	11/22/2022 2:56 AM	File folder
tests	11/22/2022 2:56 AM	File folder
vendor	12/4/2022 7:40 PM	File folder
.editorconfig	11/22/2022 2:56 AM	Editor Config Sour...
.env	12/4/2022 7:40 PM	ENV File
.env.example	12/4/2022 7:40 PM	EXAMPLE File
.gitattributes	11/22/2022 2:56 AM	Git Attributes Sour...
.gitignore	11/22/2022 2:56 AM	Git Ignore Source ...
artisan	11/22/2022 2:56 AM	File
composer	11/22/2022 2:56 AM	JSON Source File
composer.lock	12/4/2022 7:36 PM	LOCK File
package	11/22/2022 2:56 AM	JSON Source File
phpunit	11/22/2022 2:56 AM	XML Document
README	11/22/2022 2:56 AM	Markdown Source ...
vite.config	11/22/2022 2:56 AM	JavaScript Source ...



EXAMPLE-SMITH
- app
- bootstrap
- config
- database
- lang
- public
- resources
- routes
- storage
- tests
- vendor
- .editorconfig
- .env
- .env.example
- .gitattributes
- .gitignore
- artisan
- composer.json
- composer.lock
- package.json

Start

New File...

Open File...

Open Folder...

Recent

9.0	C:\Users\sajan
smith	C:\xampp\htdocs\dash...
tcreate.php (Workspace)	C:\xa...
WT22lab (Workspace)	C:\xam...
wtppt-22	C:\Users\sajan\my-s...

More...

Walkthroughs

Learn the Funda...

Jump right into VS Code and get an overview of the must-have features.

Boost your Prod...

Get sta... Updated

More...

Laravel routes

Introduction To Laravel Routes

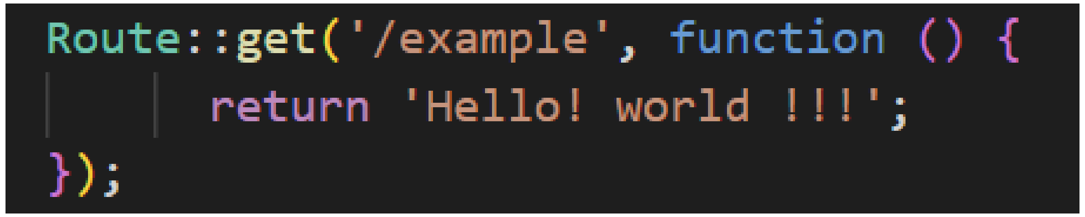
- Routing accepts the request and redirects it to the relevant controller function.
- There are two main route files in the Laravel Framework:
- routes/web.php: This file is used to register web routes.
- routes/api.php: This file is used to register API routes.
- The following code segment shows the default web route registered by Laravel to display the welcome page from views/welcome.blade.php.

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

- In the above route, Route is the class used to define the function get().
- The function get() has a parameter “/” which indicates the root URL of the Laravel application.

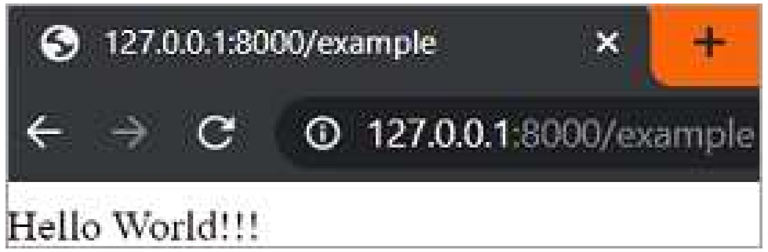
Introduction To Laravel Routes

- Creating a Route
- Step 1:
- Add the following code segment in routes/web.php file to register a new route.



```
Route::get('/example', function () {
        return 'Hello! world !!!';
});
```

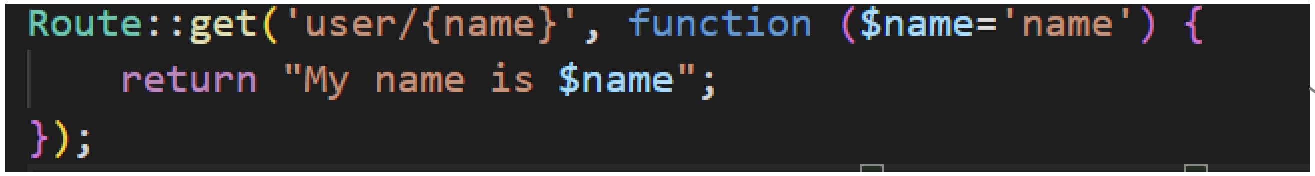
- Step 2: Visit the URL: http://127.0.0.1:8000/example to see the output.
- Step 3: The following screenshot shows the output.



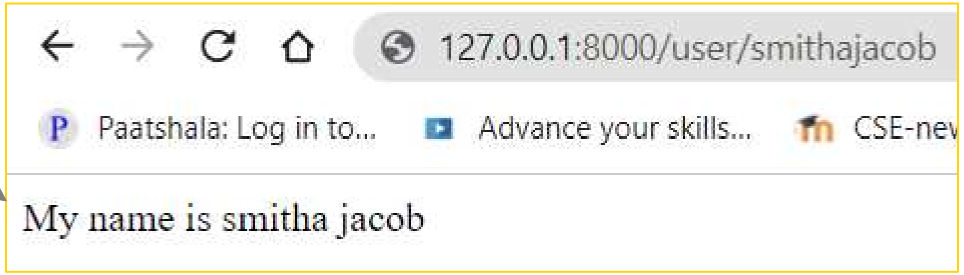
```
Hello World!!!
```

Introduction To Laravel Routes

- **Routing Parameters**
- The Laravel Framework uses two types of route parameters.
- **#1) Required parameters**
- The required parameters are the parameters that pass to the URL as shown below.



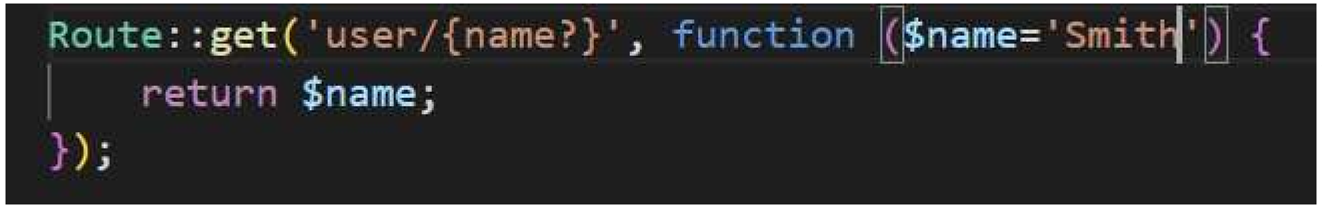
```
Route::get('user/{name}', function ($name='name') {
    return "My name is $name";
});
```



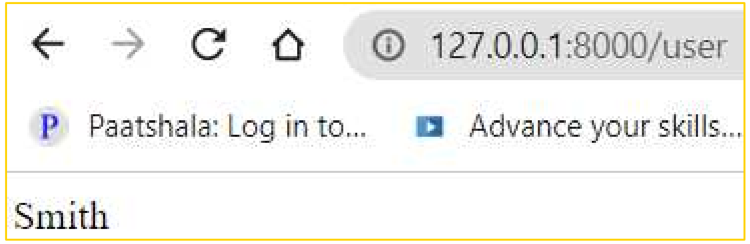
```
127.0.0.1:8000/user/smithajacob
```

My name is smitha jacob

- **#2) Optional parameters**
- Place “**?**” after the router parameter to make it **optional** as shown below.



```
Route::get('user/{name?}', function ($name='Smith') {
    return $name;
});
```

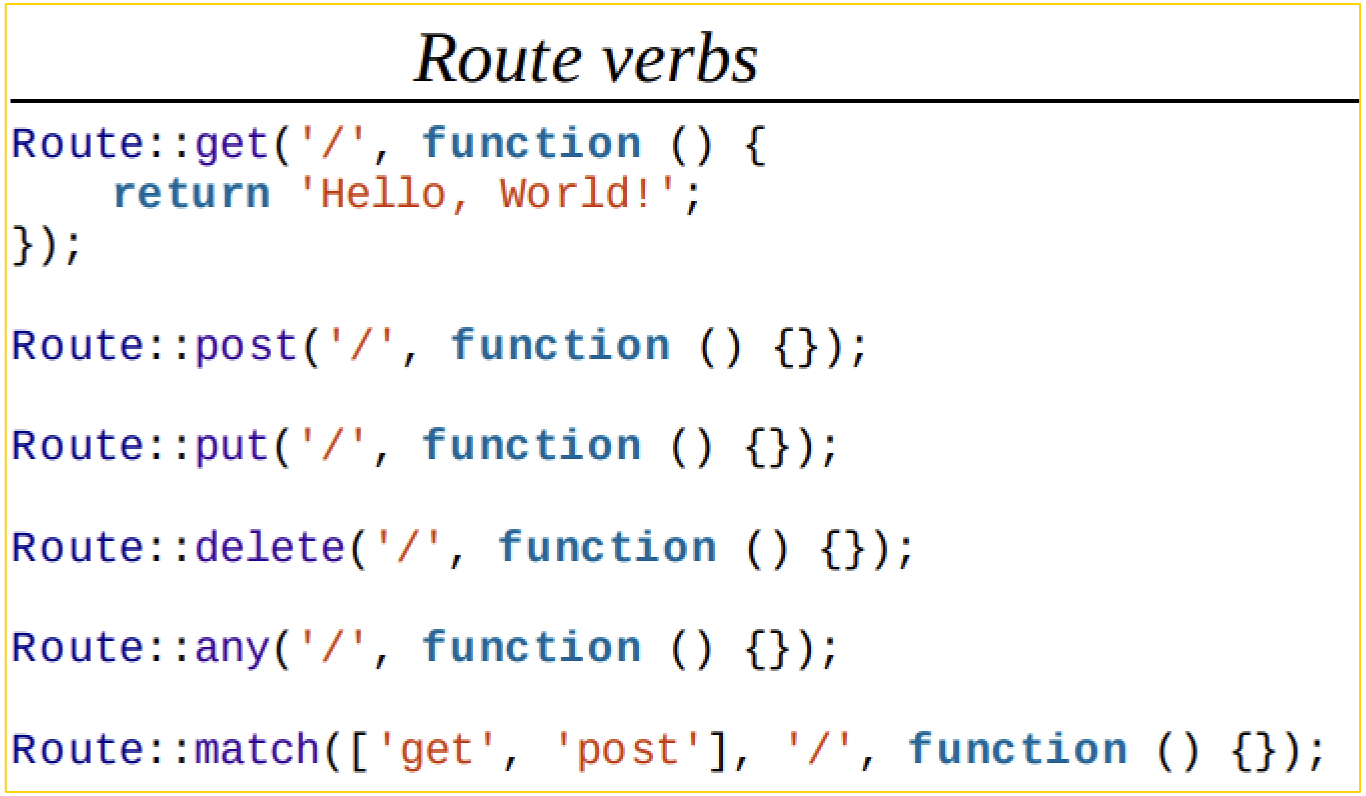


```
127.0.0.1:8000/user
```

Smith

Route verbs

- Routing accepts the request and redirects it to the relevant controller function.
- Route::get is used in our route definitions.
- This means we're telling Laravel to only match for these routes when the HTTP request uses the GET action
- The most common are GET and POST, followed by PUT, DELETE, and PATCH.
- Each method communicates a different thing to the server, and to your code, about the intentions of the caller



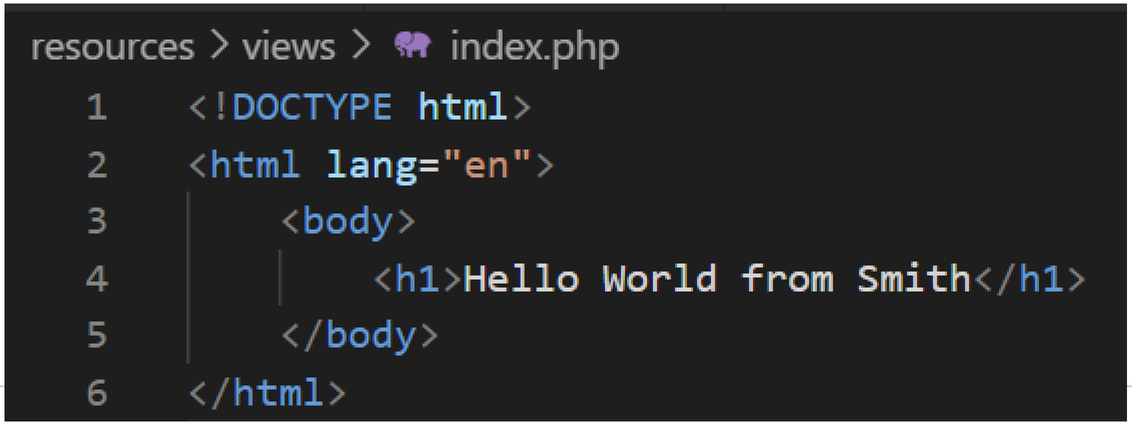
Route verbs

```
Route::get('/', function () {
    return 'Hello, World!';
});

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

Route::put('/', function () {});

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

Route::any('/', function () {});

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

Laravel Views

Introduction To Views

- In MVC architecture, the character 'V' stands for **View**. Views contain HTML, CSS and JavaScript.
- The **Views** represent the **frontend** of the Laravel application, and it is used to display the content for the user.
- All the views are stored in the **resources/views** directory. By default, the Laravel Framework provides the **welcome.blade.php** file.
- **Creating a View**
- **Step 1:** Create a view named **index.php** in the **resources/views** directory and save the following code

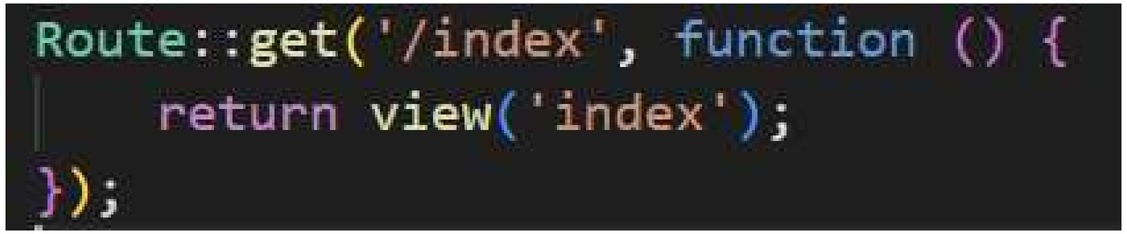


resources > views > index.php

```
<!DOCTYPE html>
<html lang="en">
    <body>
        <h1>Hello World from Smith</h1>
    </body>
</html>
```

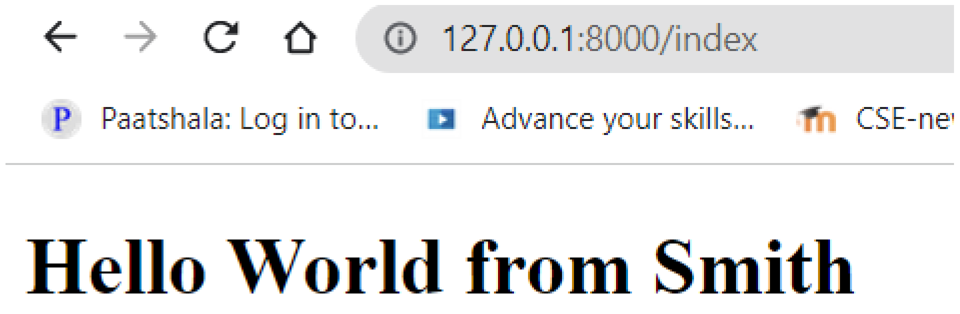
introduction To Views

- **Step 2:** Add the following code segment in the **routes/web.php** file to register a new route.



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

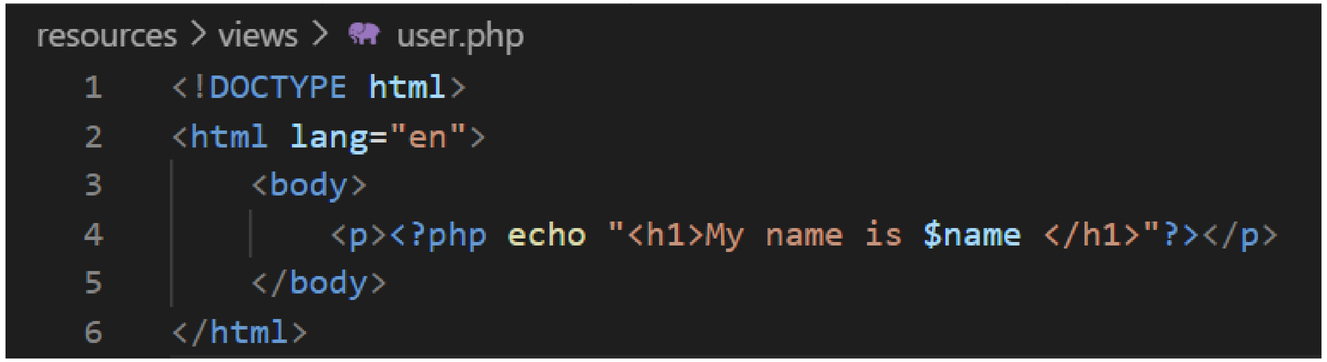
- **Step 3:** Visit the **URL: http://127.0.0.1:8000/index** to see the output.
- **Step 4:** The following screenshot shows the output.



Hello World from Smith

Passing Data to Views

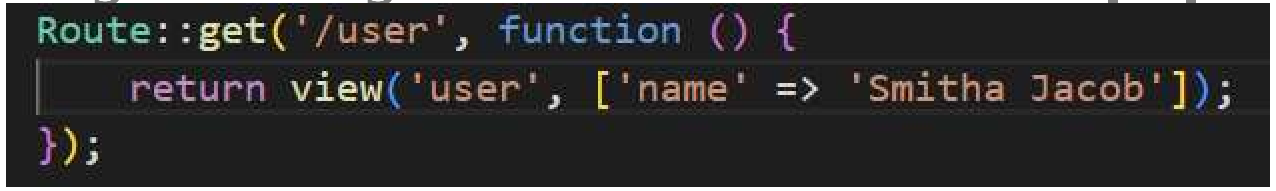
- **Step 1:** Create a view named **user.php** in the **resources/views** directory and save the following code.



resources > views > user.php

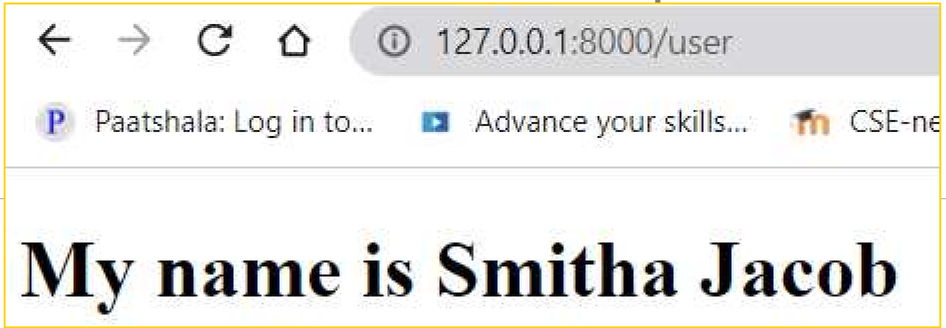
```
<!DOCTYPE html>
<html lang="en">
    <body>
        <p><?php echo "<h1>My name is $name </h1>"?></p>
    </body>
</html>
```

- **Step 2:** Add the following code segment in the **routes/web.php** file to register a new route.



```
Route::get('/user', function () {
    return view('user', ['name' => 'Smitha Jacob']);
});
```

- **Step 3:** Visit the **URL: http://127.0.0.1:8000/user** to see the output.
- **Step 4:** The following screenshot shows the output.

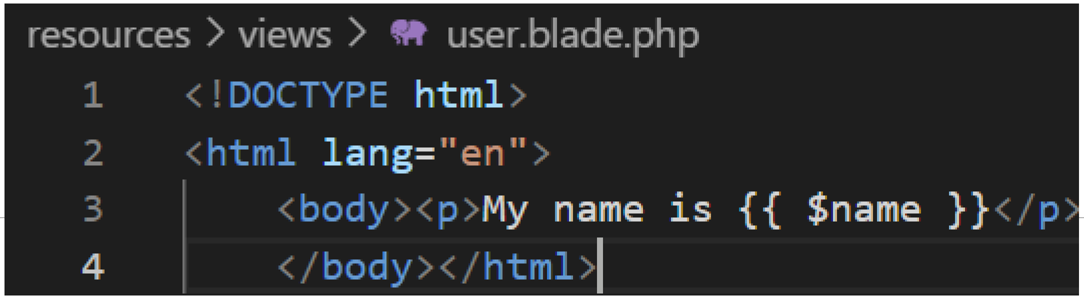


127.0.0.1:8000/user

My name is Smitha Jacob

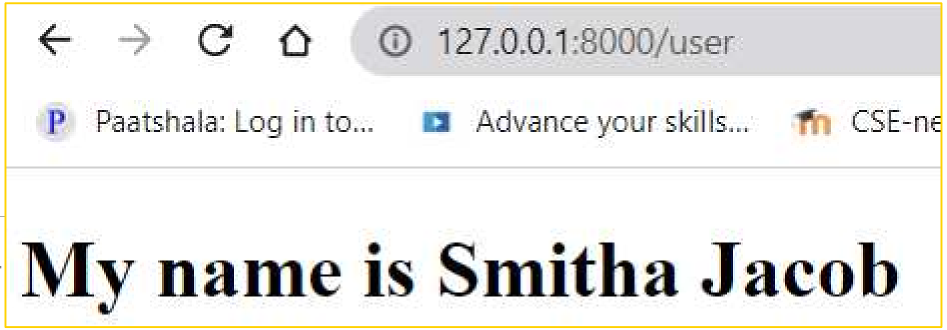
Introduction to blade template

- Blade templates use **.blade.php** extension. The blade templates stored in the **resources/views** directory.
- **Usage of Blade Templates**
- **Usage 1: variable values**
- In blade templates, we can use **{{ $variable }}** to print the value of a variable instead of **<?php echo $variable ?>** As you see, there is **no** need to write **PHP tags** or the **echo** keyword.
- **Step 1:** Rename the file **user.php** to **user.blade.php** (in the **resources/views** directory) and modify the existing code as shown below.
- **Step 2:** Visit the **URL: http://127.0.0.1:8000/user**.



resources > views > user.blade.php

```
<!DOCTYPE html>
<html lang="en">
    <body><p>My name is {{ $name }}</p>
    </body></html>
```



127.0.0.1:8000/user

My name is Smitha Jacob

Introduction to blade template

- **Usage 2: ternary operators**
- In blade templates, we can use **{{ $variable or ‘default_value’ }}** instead of **<?phpisset($variable) ? $variable : ?default_value? ?>** to write ternary operators.

Laravel controllers

Laravel Controllers

- In MVC architecture, the character 'C' stands for Controller.
- A controller communicates with the relevant model if necessary and loads the view to display the content for the user.
- All the controllers are stored in the app/Http/Controllers directory.
- Creating a Controller
- Run the following command in the command prompt to create a controller named UserContoller.

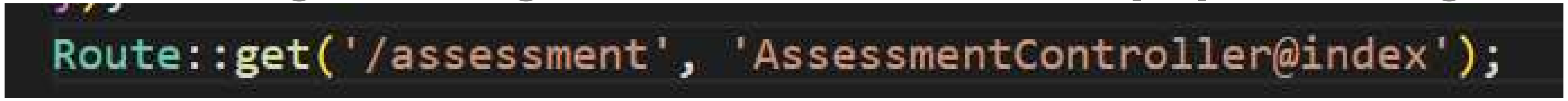
```
php artisan make:controller UserController
```

- This command will create a file named UserController.php in the app/Http/Controllers directory. By default, the Controller.php file is included with the Laravel Framework.
- Calling Controllers from Routes
- The following syntax can be used to call controllers from routes.

```
Route::get('base URI','controller@method');
```

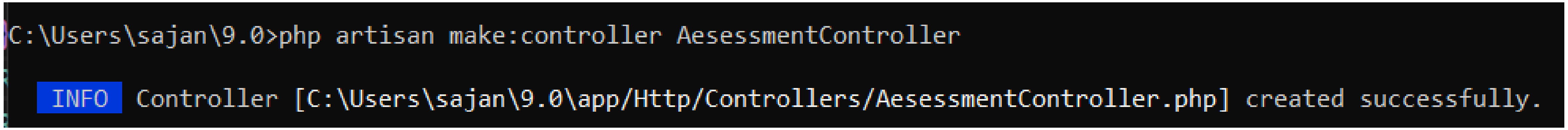
Laravel Controllers

- Let's see an example.
- **Step 1:** Add the following code segment in the **routes/web.php** file to register a new route.



```
Route::get('/assessment', 'AssessmentController@index');
```

- **Step 2:** Run the following command in the command prompt to create a controller named **AssessmentController**

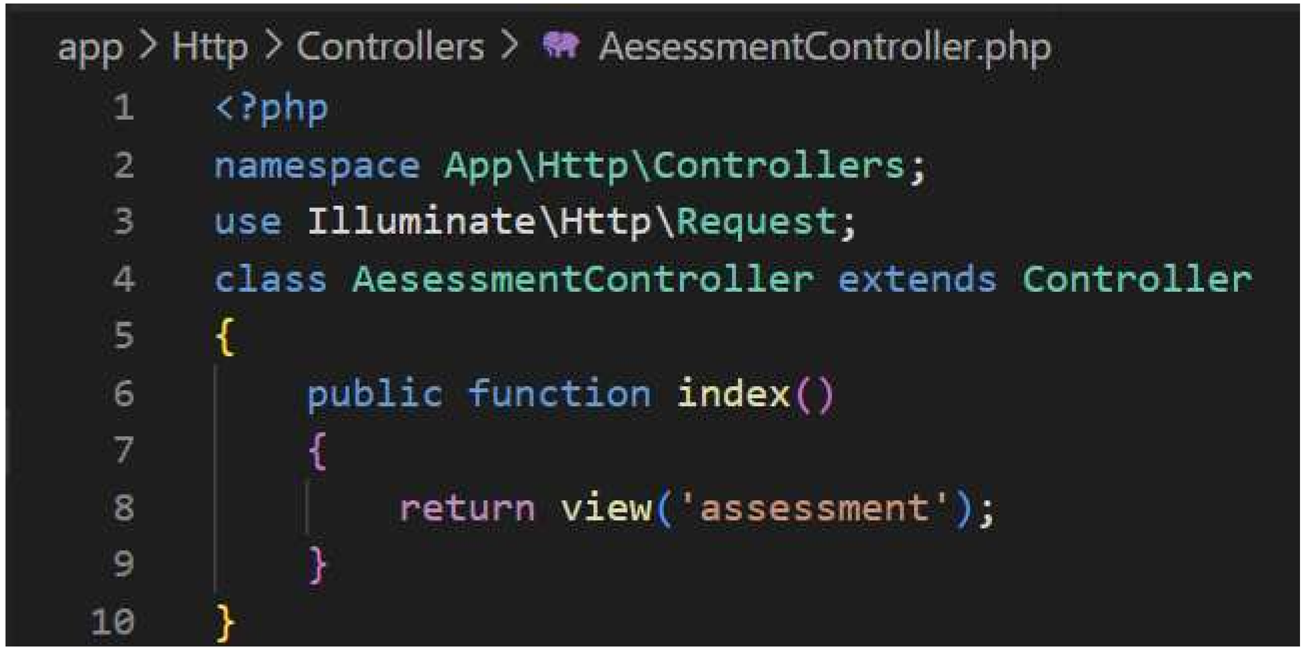


```
C:\Users\sajan\9.0>php artisan make:controller AesessmentController

  INFO  Controller [C:\Users\sajan\9.0\app/Http/Controllers/AesessmentController.php] created successfully.
```

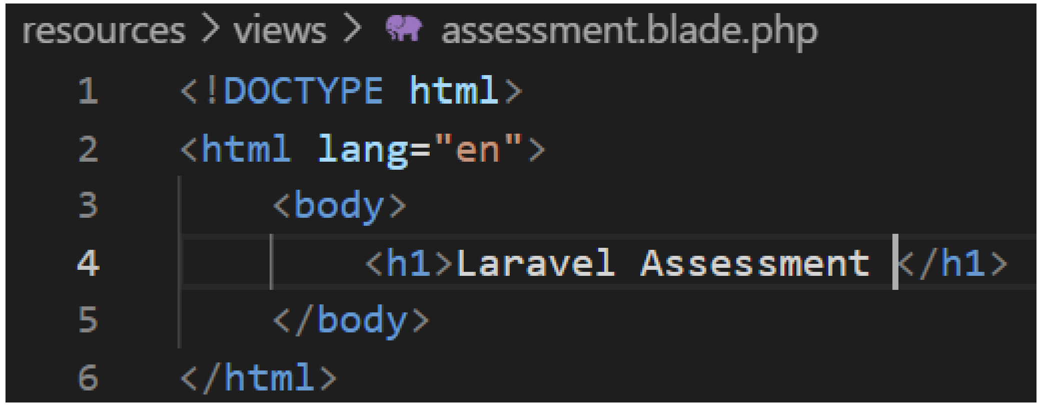
- This command will create a file named **AssessementController.php** in the **app/Http/Controllers** directory.
- **Step 3:** Create the **index** function in the **AssessementController** as shown below

Laravel Controllers

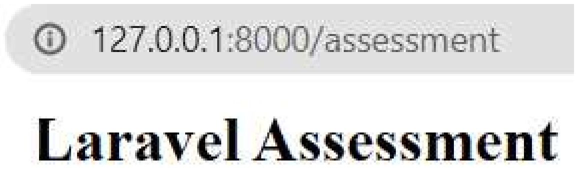
- **Step 4:** Create a view named **assessment.php** in the **resources/views** directory and save the following code.
- **Step 5:** Visit the **URL:** http://127.0.0.1:8000/assessment to see the output.
- **Step 6:** The following screenshot shows the output.


app > Http > Controllers > AesessmentController.php

```php
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class AesessmentController extends Controller
{
    public function index()
    {
        return view('assessment');
    }
}
```


resources > views > assessment.blade.php

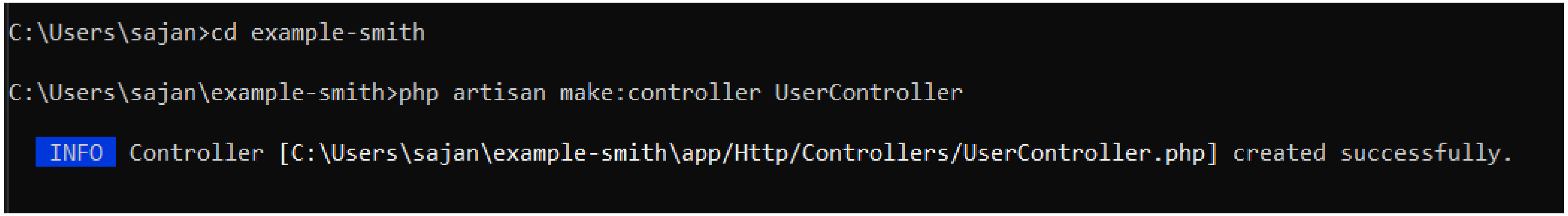
```html
<!DOCTYPE html>
<html lang="en">
    <body>
        <h1>Laravel Assessment </h1>
    </body>
</html>
```


127.0.0.1:8000/assessment

Laravel Assessment

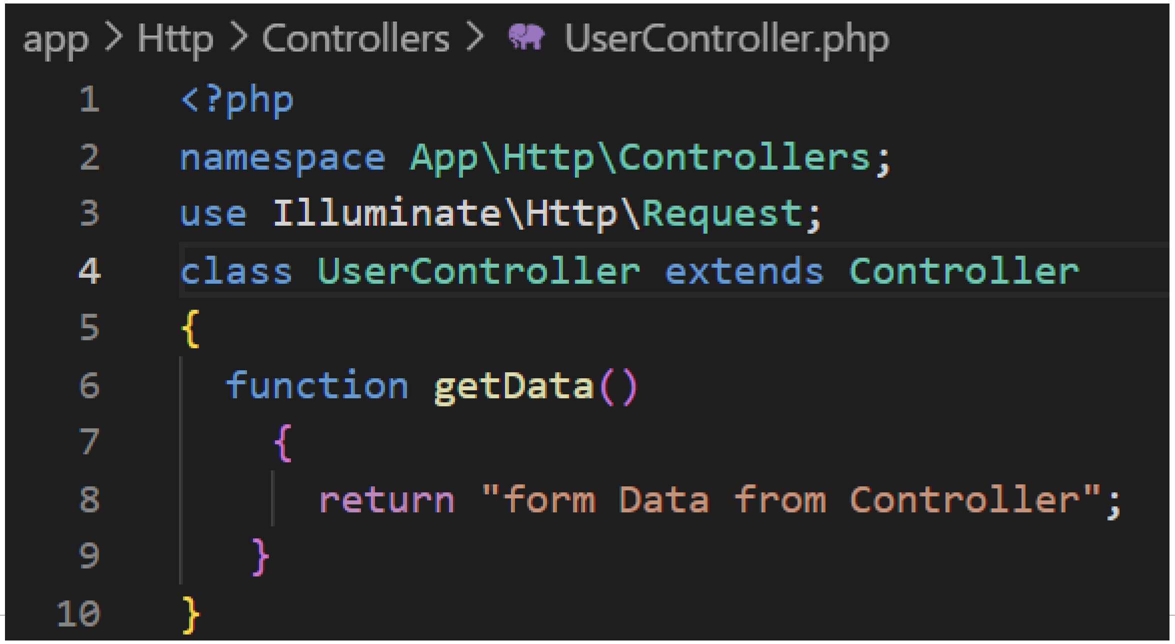
Login Example using Laravel Controllers

- **Step 1:**Run the command prompt and make the controller named UserController



```
C:\Users\sajan>cd example-smith

C:\Users\sajan\example-smith>php artisan make:controller UserController

   INFO  Controller [C:\Users\sajan\example-smith\app/Http/Controllers/UserController.php] created successfully.
```

- Step 2: Add the following code segment in the http**/UserController.php** .

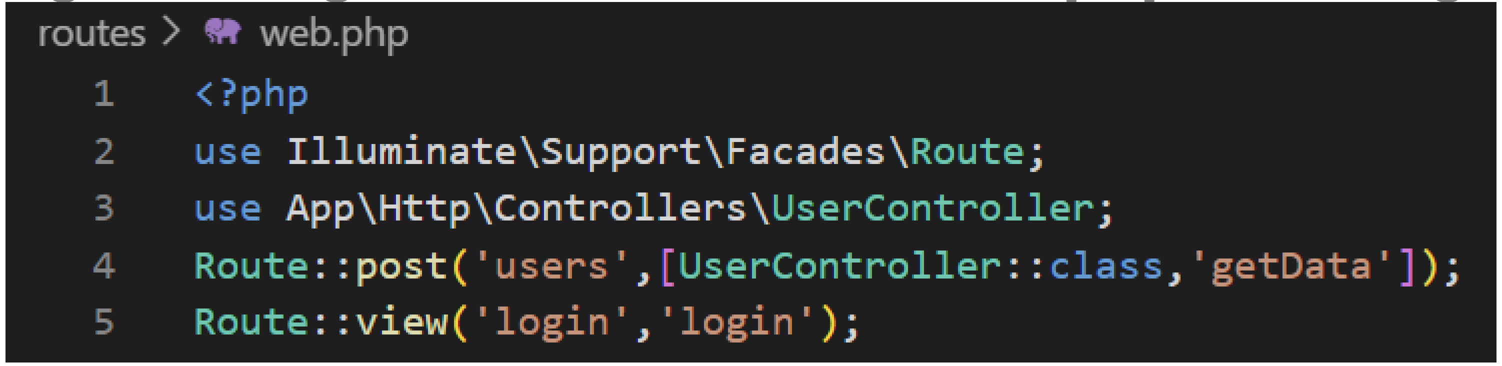


app > Http > Controllers > UserController.php

```php
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class UserController extends Controller
{
  function getData()
    {
      return "form Data from Controller";
    }
}
```

Login Example using Laravel Controllers

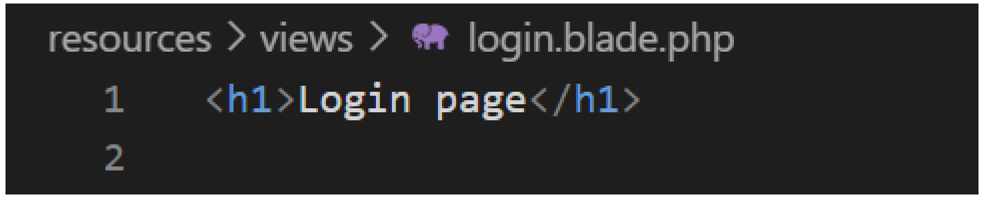
- **Step 3:**Add the following code segment in the **routes/web.php** file to register a new route.



routes > web.php

```php
<?php
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\UserController;
Route::post('users',[UserController::class,'getData']);
Route::view('login','login');
```

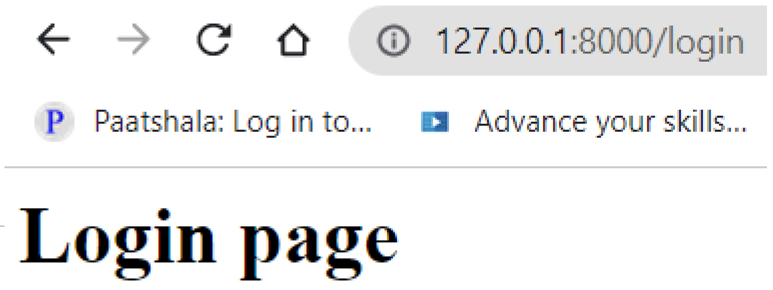
- **Step 4**:create a view named login.blade.php in **resources/views/login.blade.php**



resources > views > login.blade.php

```html
<h1>Login page</h1>

```

- **Step5:**Visit the **URL:** http://127.0.0.1:8000/login to see the output.
- Step 6: The following screenshot shows the output



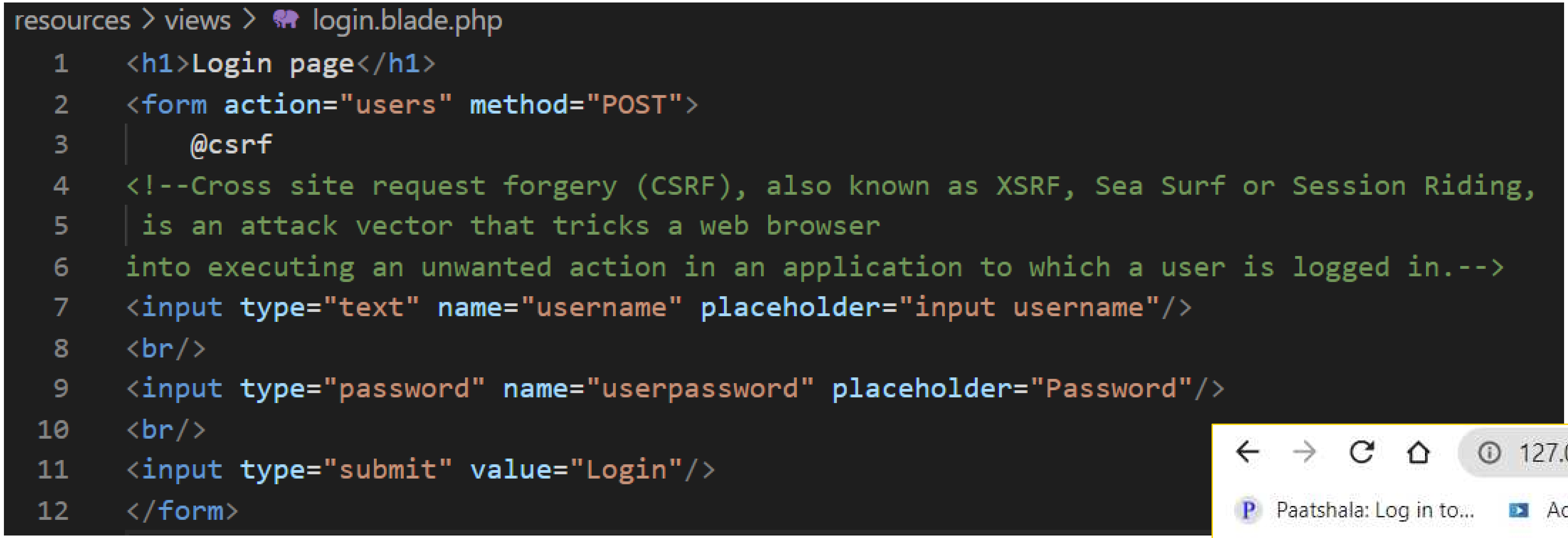
127.0.0.1:8000/login

Paatshala: Log in to... Advance your skills...

Login page

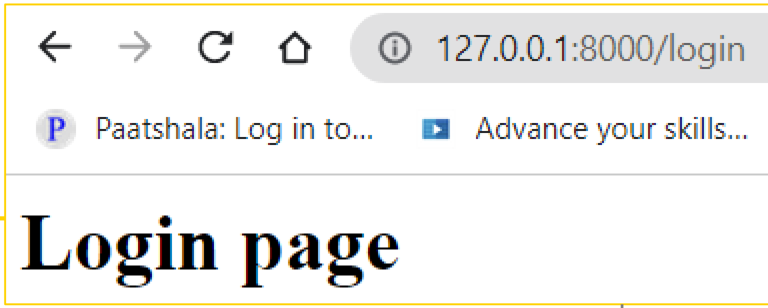
Login Example using Laravel Controllers

- Modify the Login Page



resources > views > login.blade.php

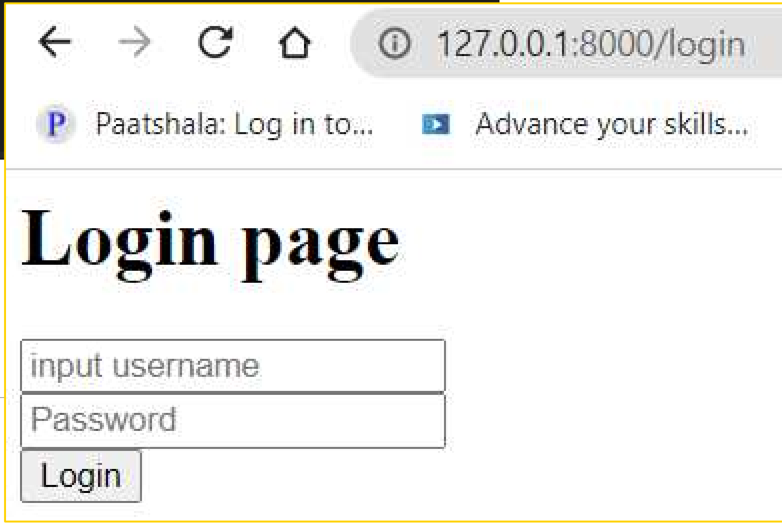
```
<h1>Login page</h1>
<form action="users" method="POST">
    @csrf
<!--Cross site request forgery (CSRF), also known as XSRF, Sea Surf or Session Riding,
 is an attack vector that tricks a web browser
into executing an unwanted action in an application to which a user is logged in.-->
<input type="text" name="username" placeholder="input username"/>
<br/>
<input type="password" name="userpassword" placeholder="Password"/>
<br/>
<input type="submit" value="Login"/>
</form>
```



127.0.0.1:8000/login

Paatshala: Log in to... Advance your skills...

Login page



127.0.0.1:8000/login

Paatshala: Log in to... Advance your skills...

Login page

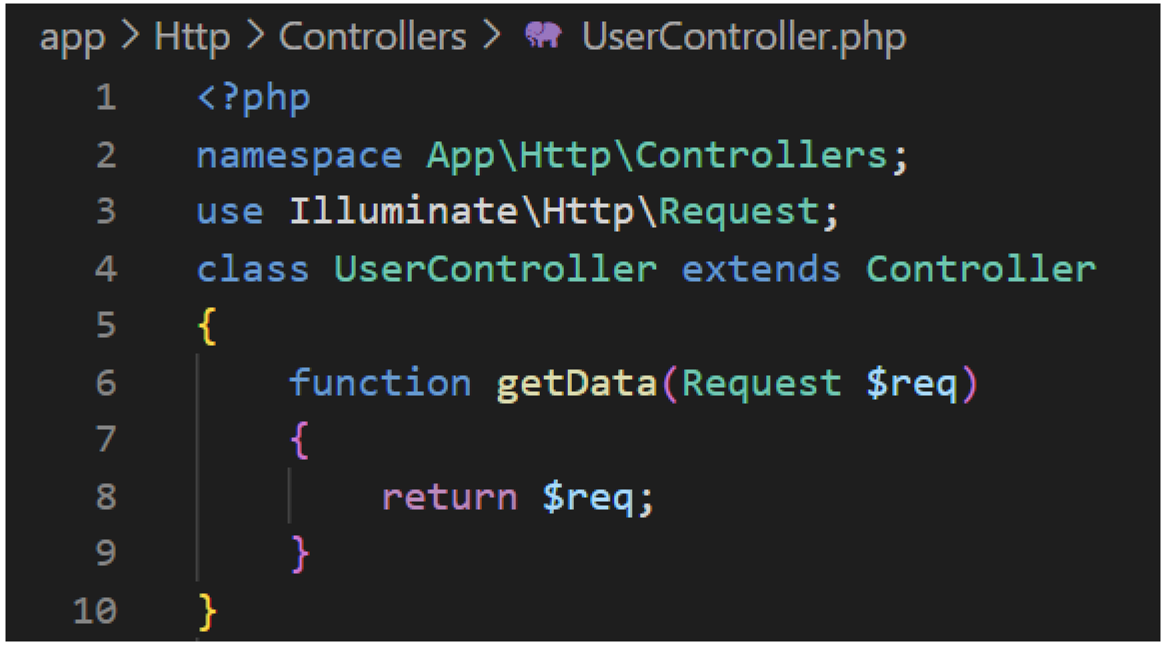
input username

Password

Login

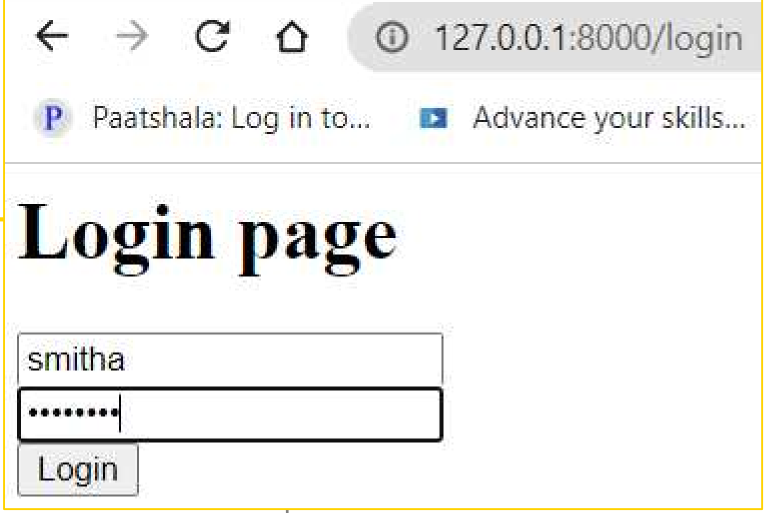
Login Example using Laravel Controllers

- Modify the UserController Page

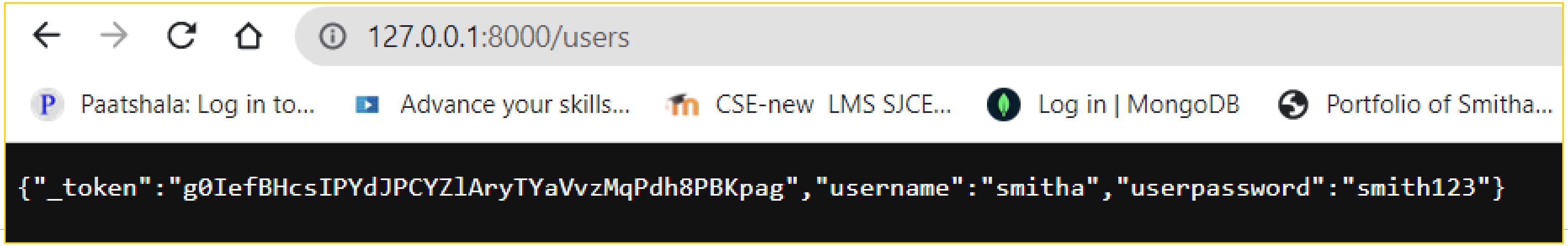


app > Http > Controllers > UserController.php

```php
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class UserController extends Controller
{
    function getData(Request $req)
    {
        return $req;
    }
}
```

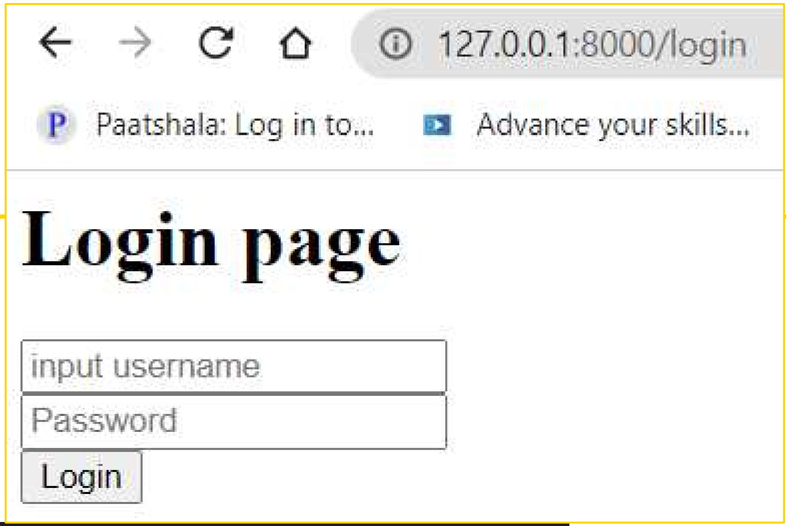


Login page



```
{"_token":"g0IefBHcsIPYdJPCYZlAryTYaVvzMqPdh8PBKpag","username":"smitha","userpassword":"smith123"}
```

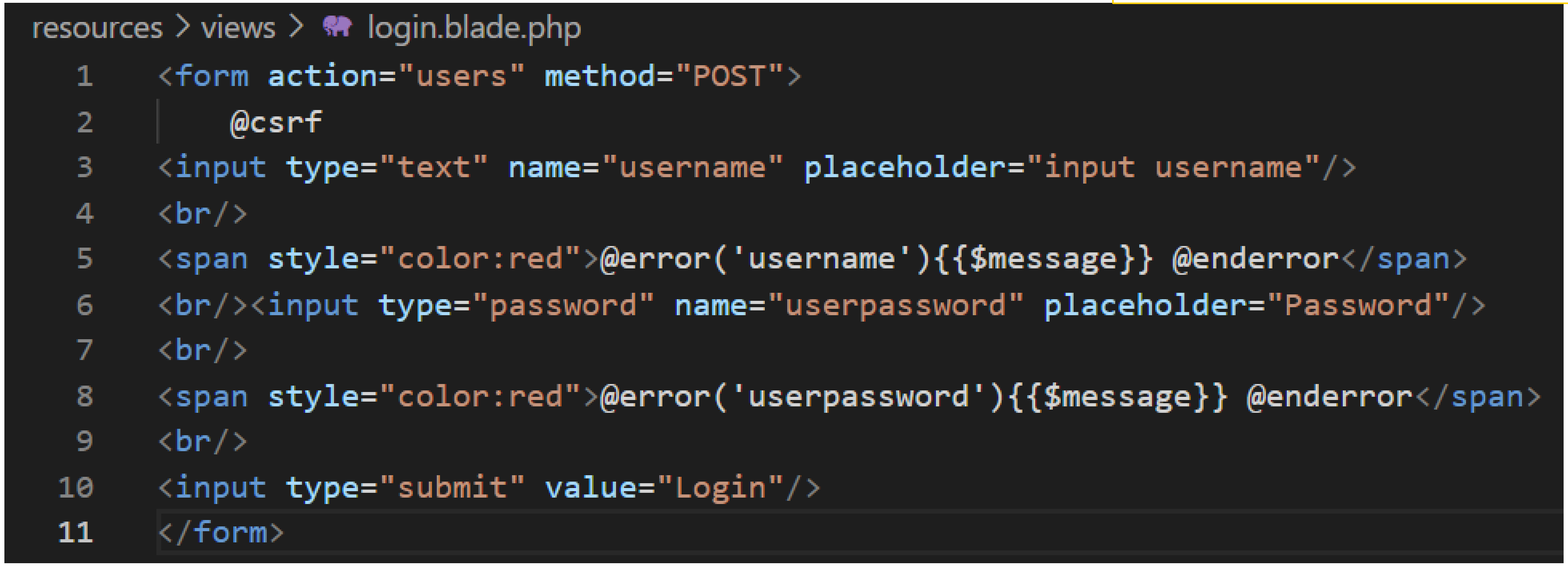
Login Example using Laravel Controllers


127.0.0.1:8000/login

Paatshala: Log in to... Advance your skills...

Login page

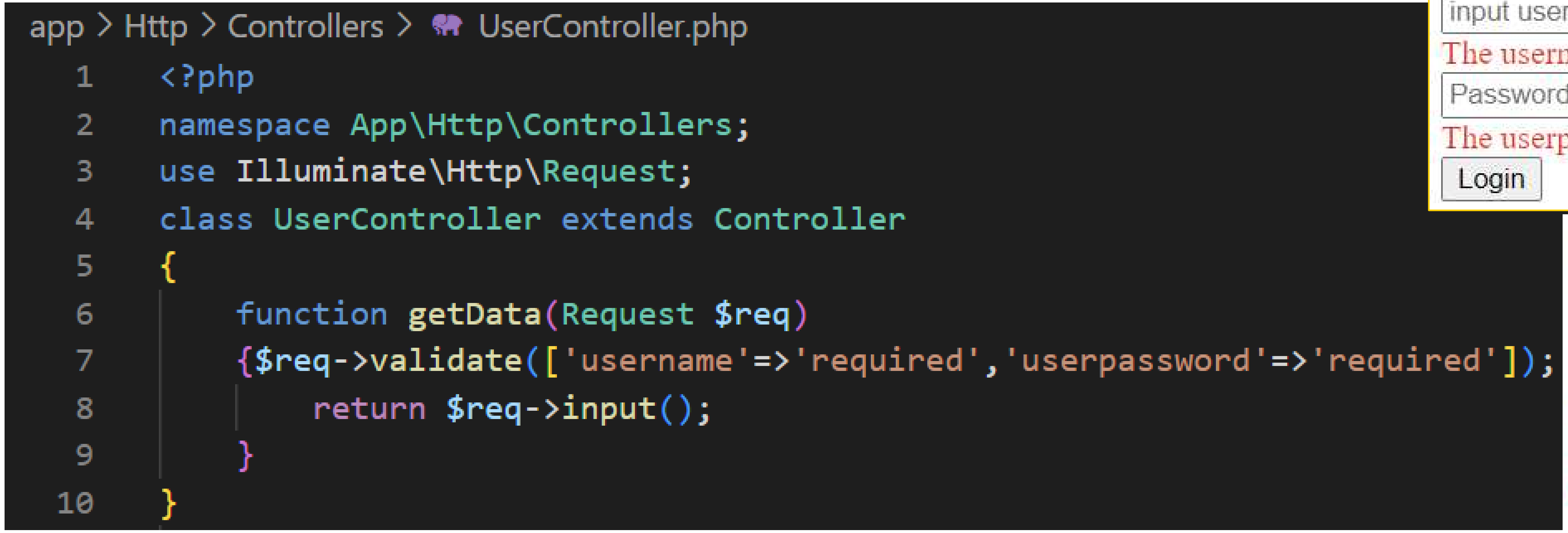
- Modify the Login Page for Validation


resources > views > login.blade.php

```
<form action="users" method="POST">
    @csrf
<input type="text" name="username" placeholder="input username"/>
<br/>
<span style="color:red">@error('username'){{$message}} @enderror</span>
<br/><input type="password" name="userpassword" placeholder="Password"/>
<br/>
<span style="color:red">@error('userpassword'){{$message}} @enderror</span>
<br/>
<input type="submit" value="Login"/>
</form>
```

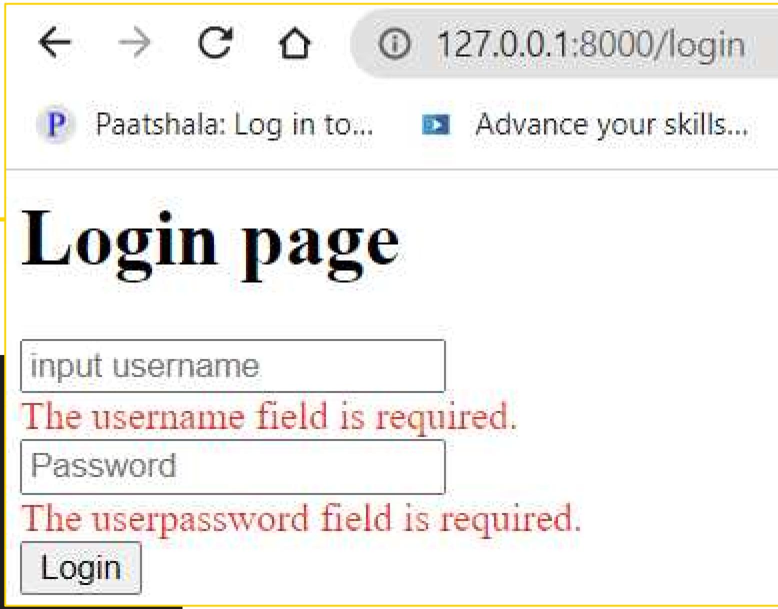
Login Example using Laravel Controllers

- Modify the UserController Page



app > Http > Controllers > UserController.php

```php
<?php
namespace App\Http\Controllers;
use Illuminate\Http\Request;
class UserController extends Controller
{
    function getData(Request $req)
    {$req->validate(['username'=>'required','userpassword'=>'required']);
        return $req->input();
    }
}
```



127.0.0.1:8000/login

Paatshala: Log in to... Advance your skills...

Login page

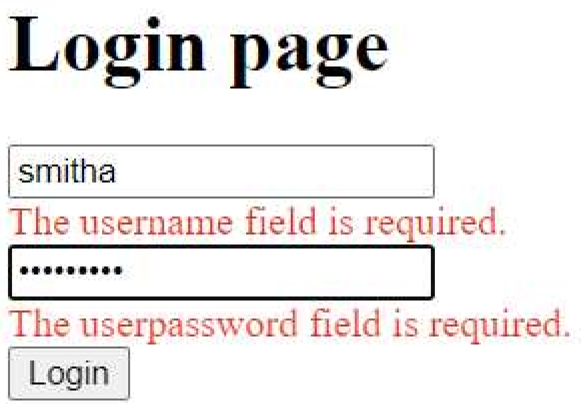
input username

The username field is required.

Password

The userpassword field is required.

Login



Login page

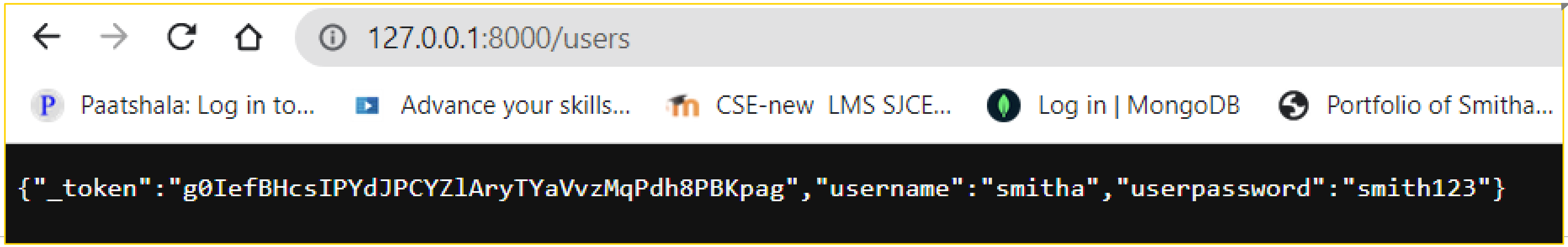
smitha

The username field is required.

•••••••••

The userpassword field is required.

Login



127.0.0.1:8000/users

Paatshala: Log in to... Advance your skills... CSE-new LMS SJCE... Log in | MongoDB Portfolio of Smitha...

```
{"_token":"g0IefBHcsIPYdJPCYZlAryTYaVvzMqPdh8PBKpag","username":"smitha","userpassword":"smith123"}
```

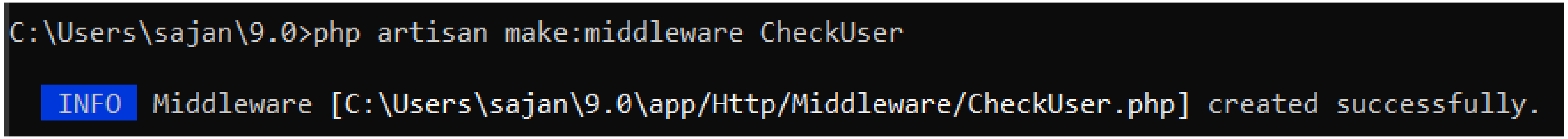
Laravel Middleware

Middleware

- **What Is Middleware**
- In simple words, middleware is a bridge between a request and a response.
- It provides a mechanism to filter **HTTP** requests.
- All the middleware files are stored in the **app/Http/Middleware** directory.
- The Laravel Framework has a middleware to check whether the user of the Laravel application is authenticated or not.
- If the **user is authenticated**, it directs to the **home page** and if the **user is not authenticated**, it **redirects to the login page**.
- There are a lot of wrappers around Laravel's request and response cycle, including something called middleware.
- When your route closure or controller method is done , it's not time to send the output to the browser yet; returning the content allows it to continue flowing through the response stack and the middleware before it is returned back to the user.

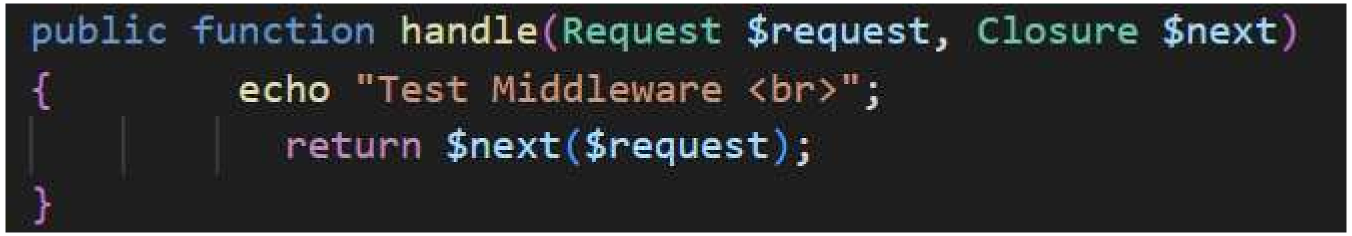
Middleware

- **Creating a Middleware**
- **Step 1:** Run the following command in the command prompt to create a middleware called **CheckUser**.



```
C:\Users\sajan\9.0>php artisan make:middleware CheckUser

  INFO  Middleware [C:\Users\sajan\9.0\app/Http/Middleware/CheckUser.php] created successfully.
```

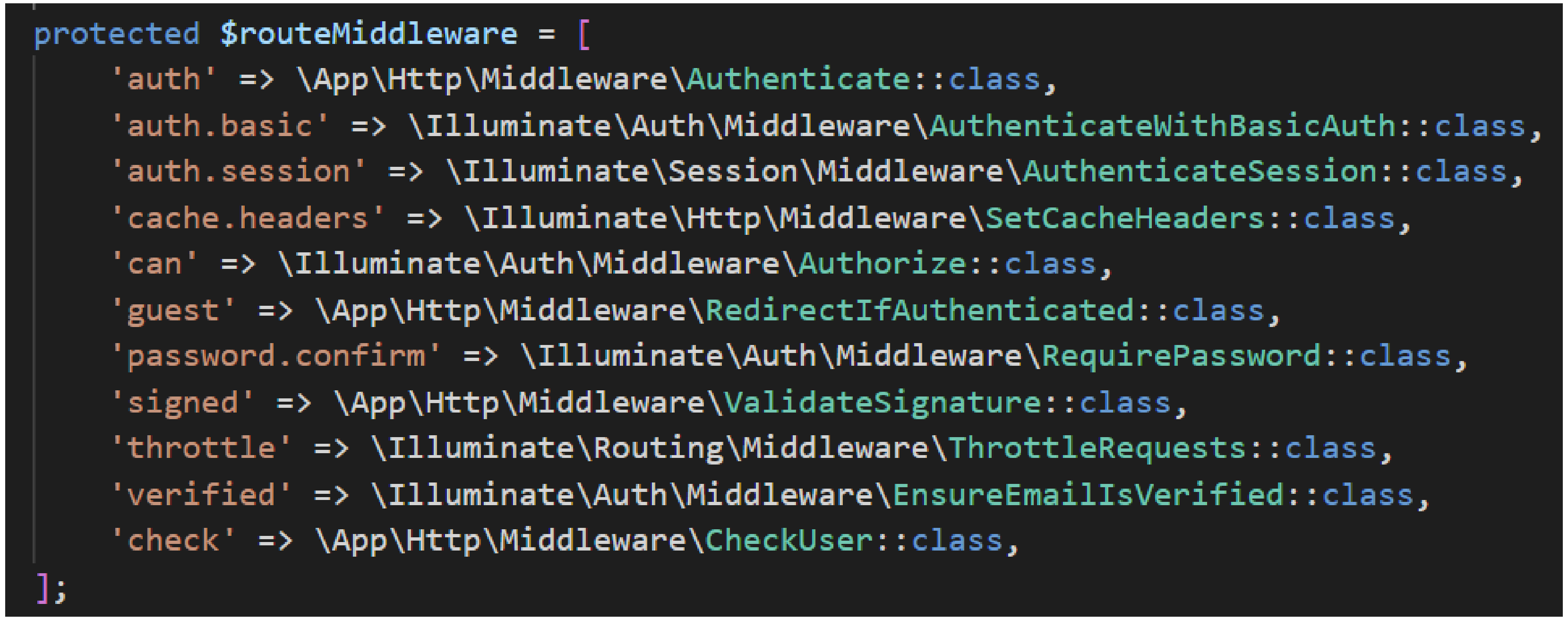
- This command will create a file named **CheckUser.php** in the **app/Http/Middleware** directory.
- **Step 2:** Open the **CheckUser.php** file and modify the existing code as shown below in class CheckUser.



```
public function handle(Request $request, Closure $next)
{         echo "Test Middleware <br>";
            return $next($request);
}
```

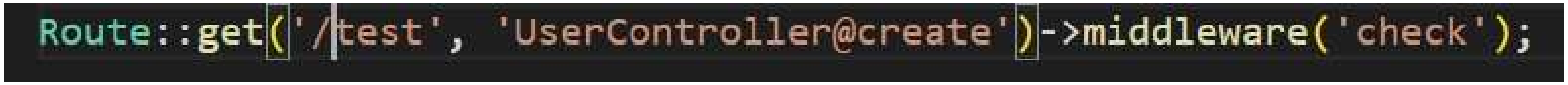
Middleware

- **Step 3:** Open the **Kernel.php** file in the **app/Http** directory and add the path of the **CheckUser** middleware as shown below.



```
protected $routeMiddleware = [
    'auth' => \App\Http\Middleware\Authenticate::class,
    'auth.basic' => \Illuminate\Auth\Middleware\AuthenticateWithBasicAuth::class,
    'auth.session' => \Illuminate\Session\Middleware\AuthenticateSession::class,
    'cache.headers' => \Illuminate\Http\Middleware\SetCacheHeaders::class,
    'can' => \Illuminate\Auth\Middleware\Authorize::class,
    'guest' => \App\Http\Middleware\RedirectIfAuthenticated::class,
    'password.confirm' => \Illuminate\Auth\Middleware\RequirePassword::class,
    'signed' => \App\Http\Middleware\ValidateSignature::class,
    'throttle' => \Illuminate\Routing\Middleware\ThrottleRequests::class,
    'verified' => \Illuminate\Auth\Middleware\EnsureEmailIsVerified::class,
    'check' => \App\Http\Middleware\CheckUser::class,
];
```

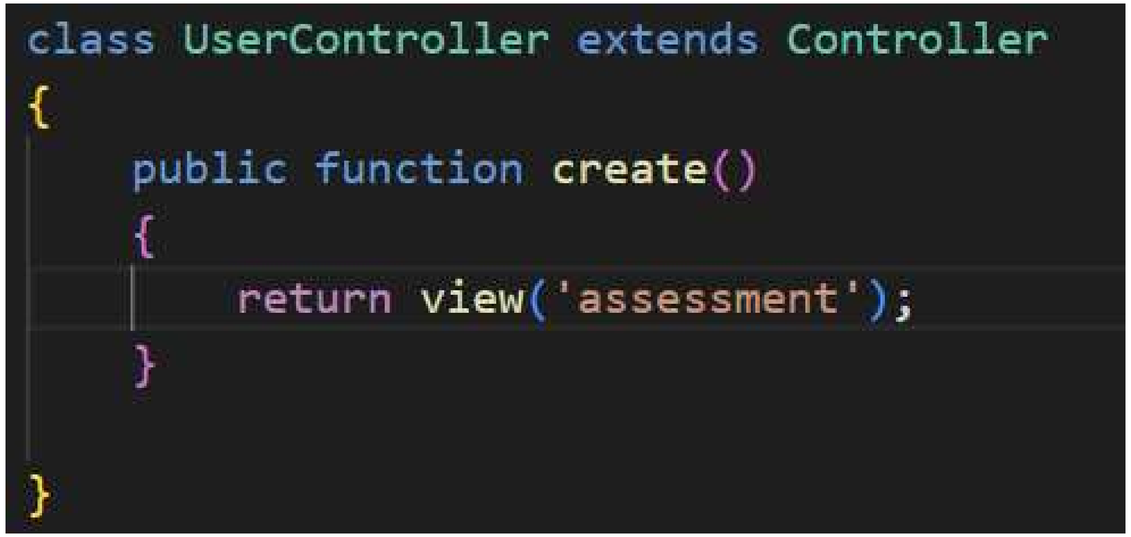
- **Step 4:** Add the following code segment in the **routes/web.php** file to register a new route.



```
Route::get('/test', 'UserController@create')->middleware('check');
```

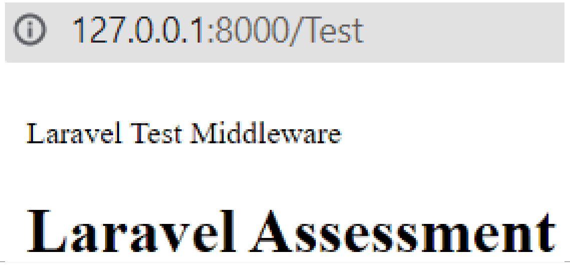
Middleware

- **Step 5:** We have already created the **User Controller**. Open the **UserController.php** file and modify the existing code as shown below.



```
class UserController extends Controller
{
    public function create()
    {
        return view('assessment');
    }

}
```

- **Step 4:** Visit the following **URL: http://127.0.0.1:8000/test** to see the output.
- **Step 5:** The following screenshot shows the output.



127.0.0.1:8000/Test

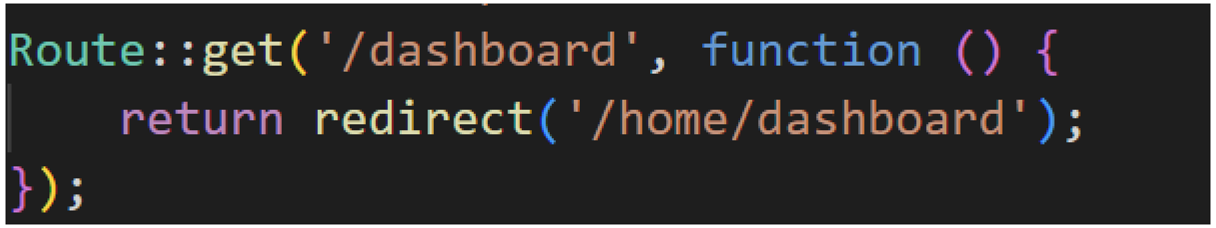
Laravel Test Middleware

Laravel Assessment

Redirection

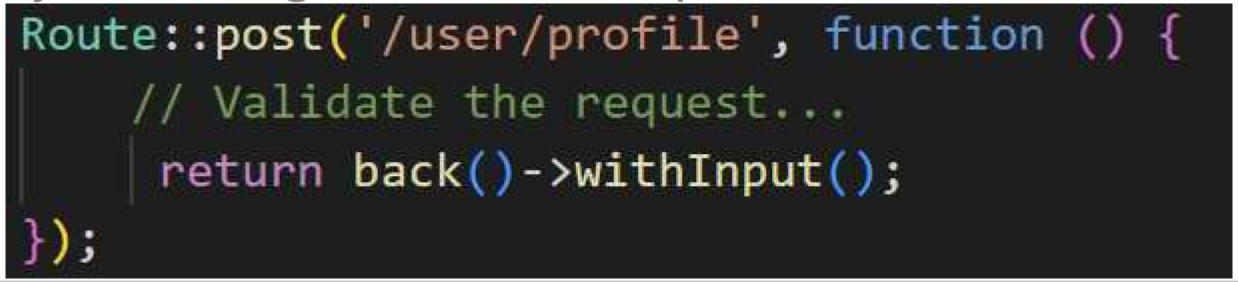
Redirection

- Creating Redirects
- Redirect responses are instances of the Illuminate\Http\RedirectResponse class, and contain the proper headers needed to redirect the user to another URL.
- There are several ways to generate a Redirect Response instance.
- The simplest method is to use the global redirect helper



```
Route::get('/dashboard', function () {
    return redirect('/home/dashboard');
});
```

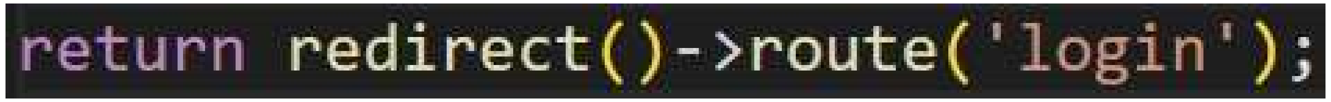
- To redirect the user to their previous location, such as when a submitted form is invalid. You may use the global back helper function.



```
Route::post('/user/profile', function () {
    // Validate the request...
    return back()->withInput();
});
```

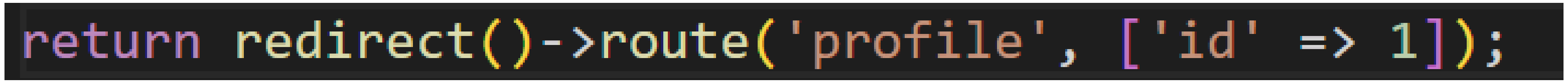
Redirection

- **Redirecting To Named Routes**
- When you call the redirect helper with no parameters, an instance of Illuminate\Routing\Redirector is returned, allowing you to call any method on the Redirector instance.
- For example, to generate a RedirectResponse to a named route, you may use the route method:



```
return redirect()->route('login');
```

- If your route has parameters, you may pass them as the second argument to the route method:
- // For a route with the following URI: profile/{id}

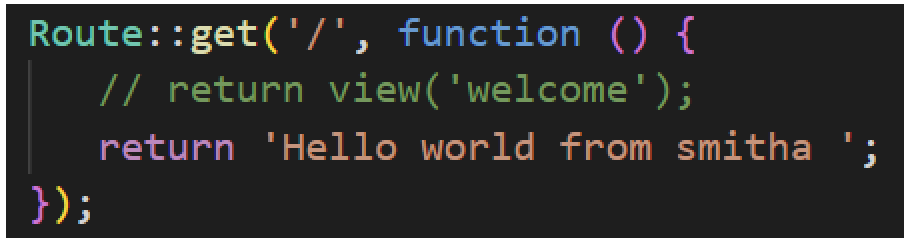


```
return redirect()->route('profile', ['id' => 1]);
```

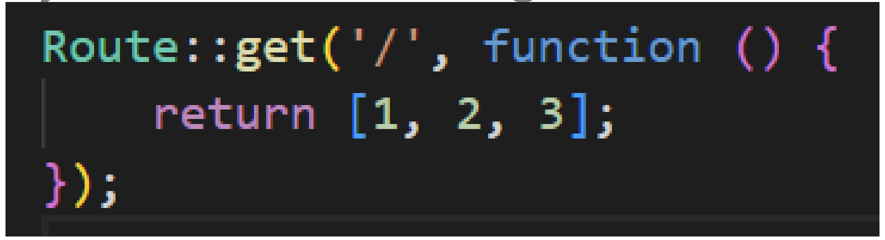
Request & Response

Response

- A web application responds to a user's request in many ways depending on many parameters
- Laravel provides several different ways to return response.
- Response can be sent either from route or from controller.
- The basic response that can be sent is simple string
- Creating Responses
- Strings & Arrays
- All routes and controllers should return a response to be sent back to the user's browser.
- Laravel framework will automatically convert the string into a full HTTP response:



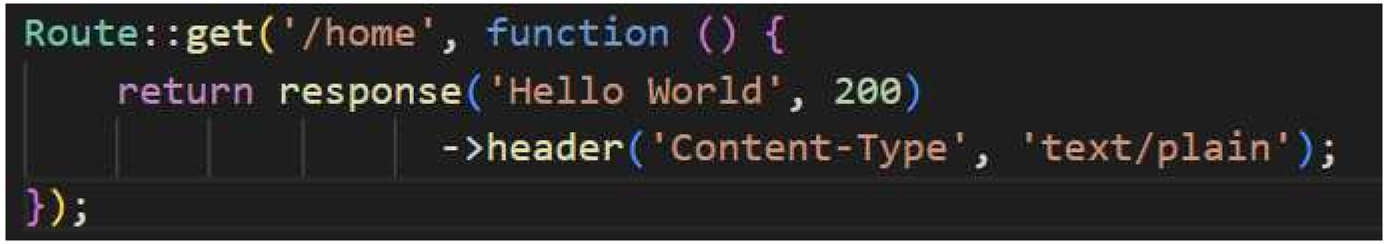
```
Route::get('/', function () {
    // return view('welcome');
    return 'Hello world from smitha ';
});
```



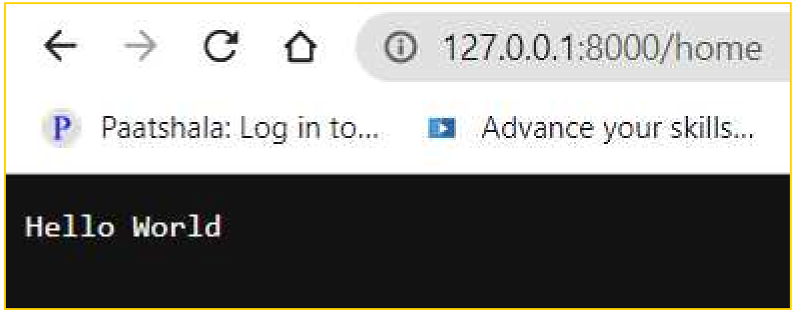
```
Route::get('/', function () {
    return [1, 2, 3];
});
```

Response

- Response Objects
- Typically, instead of returning simple strings or arrays from your route actions., we will be returning full Illuminate\Http\Response instances or views.
- Returning a full Response instance allows you to customize the response's HTTP status code and headers.
- A Response instance inherits from the Symfony\Component\HttpFoundation\Response class, which provides a variety of methods for building HTTP responses:



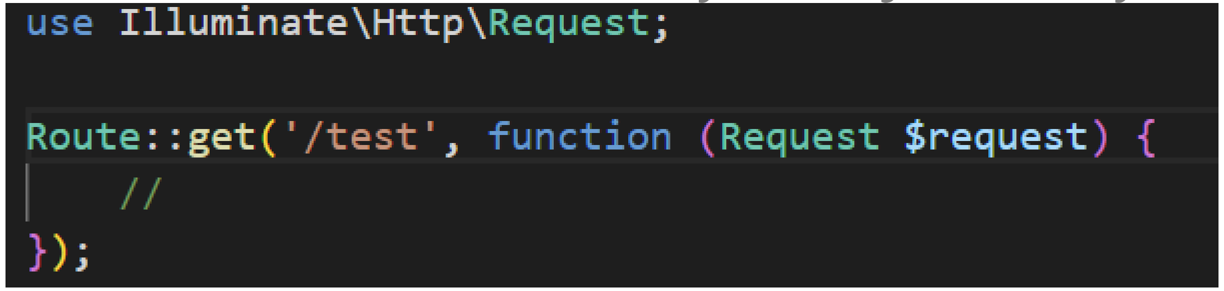
```
Route::get('/home', function () {
    return response('Hello World', 200)
                  ->header('Content-Type', 'text/plain');
});
```



```
Hello World
```

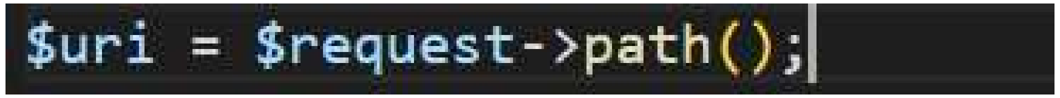
Request

- Laravel's Illuminate\Http\Request class provides an object-oriented way to interact with the current HTTP request being handled by your application as well as retrieve the input, cookies, and files that were submitted with the request.
- The incoming request instance will automatically be injected by the Laravel service container:



```php
use Illuminate\Http\Request;

Route::get('/test', function (Request $request) {
    //
});
```

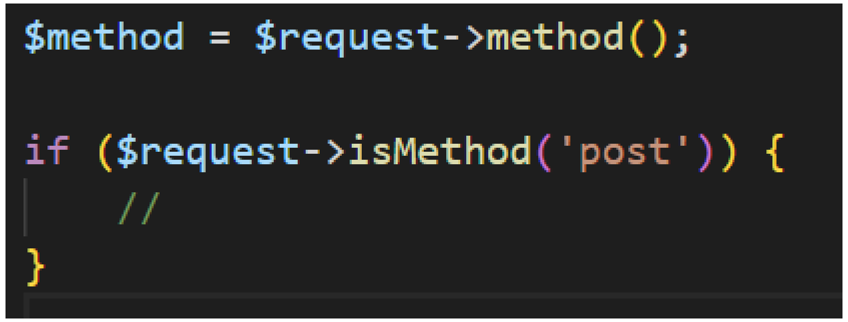
- Retrieving The Request Path:The path method returns the request's path information. So, if the incoming request is targeted at http://abc.com/foo/bar, the path method will return foo/bar:



```php
$uri = $request->path();
```

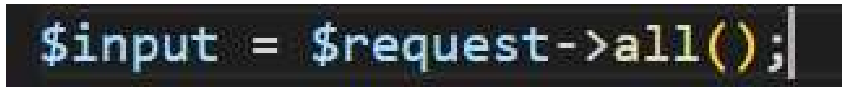
Request

- Retrieving The Request Method
- The method method will return the HTTP verb for the request. You may use the isMethod method to verify that the HTTP verb matches a given string:



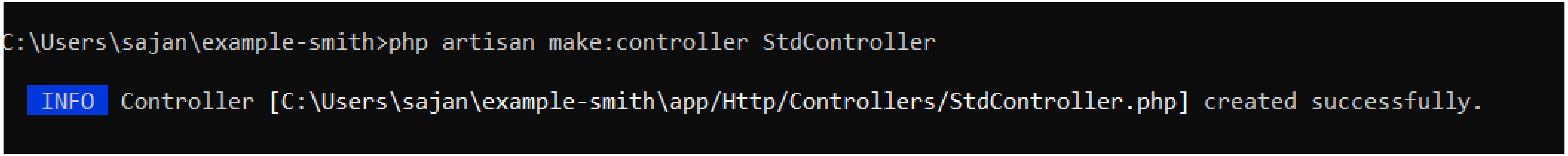
```
$method = $request->method();

if ($request->isMethod('post')) {
    //
}
```

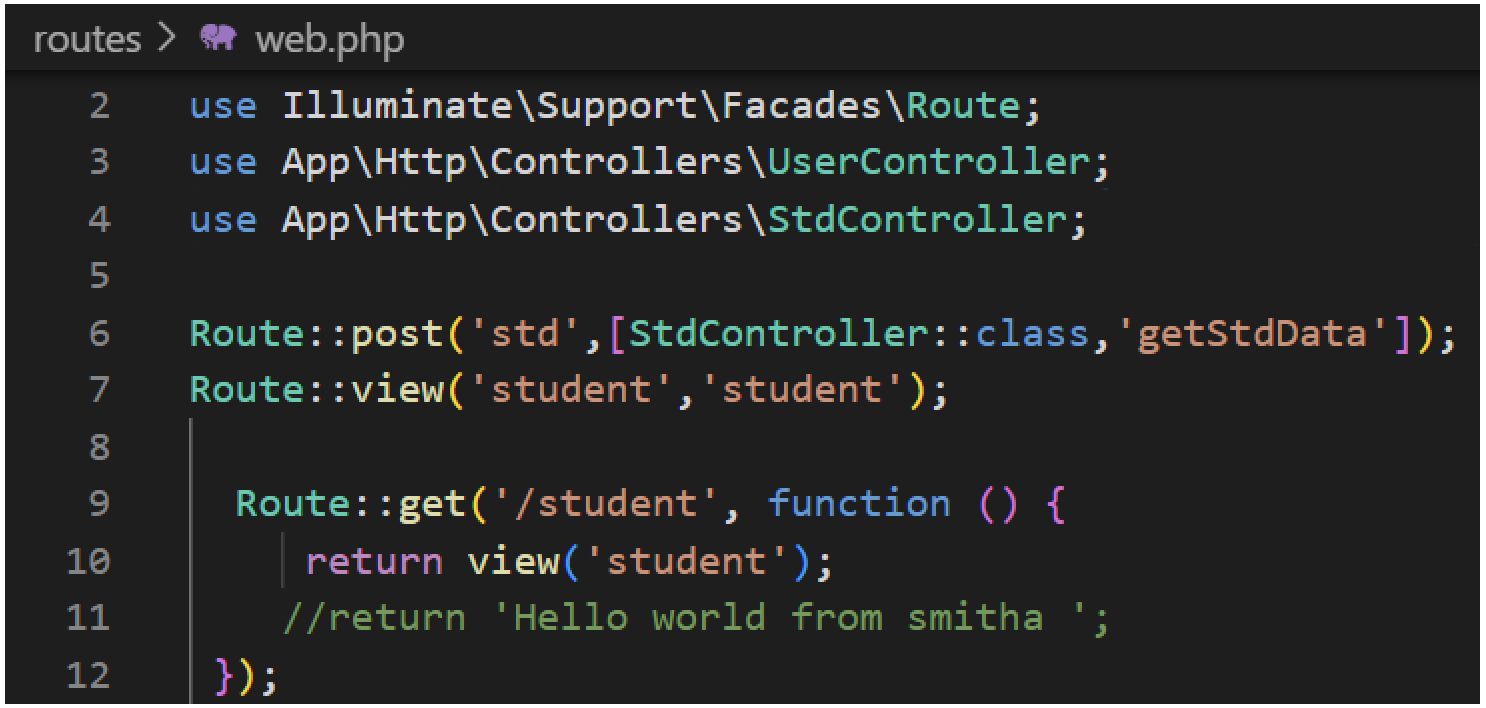
- Retrieving All Input Data
- You may retrieve all of the incoming request's input data as an array using the all method. This method may be used regardless of whether the incoming request is from an HTML form or is an XHR request:



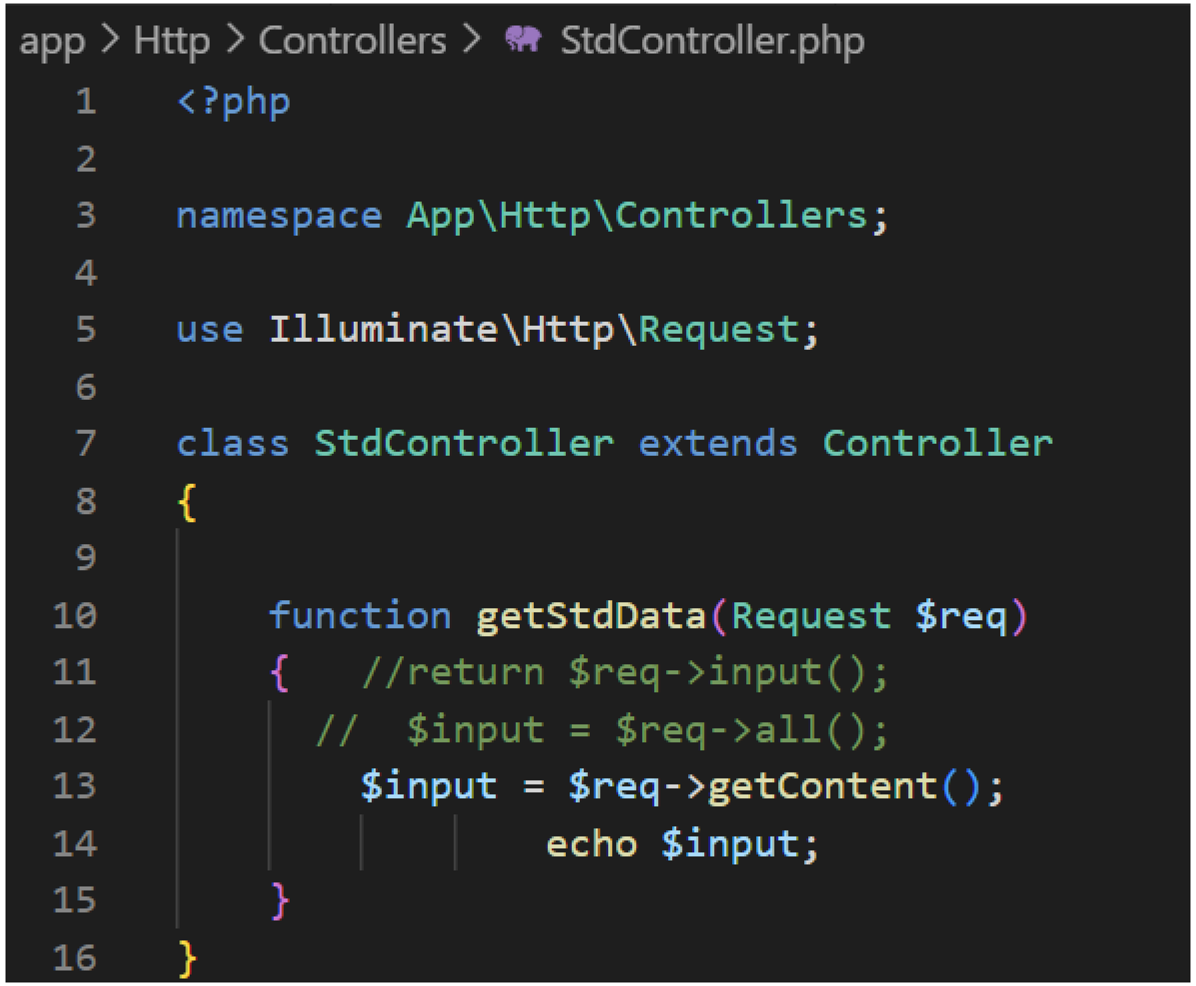
```
$input = $request->all();
```

Student Data Request and Response


```
:\Users\sajan\example-smith>php artisan make:controller StdController

  INFO  Controller [C:\Users\sajan\example-smith\app/Http/Controllers/StdController.php] created successfully.
```


routes > web.php
```php
use Illuminate\Support\Facades\Route;
use App\Http\Controllers\UserController;
use App\Http\Controllers\StdController;

Route::post('std',[StdController::class,'getStdData']);
Route::view('student','student');

 Route::get('/student', function () {
    return view('student');
   //return 'Hello world from smitha ';
});
```

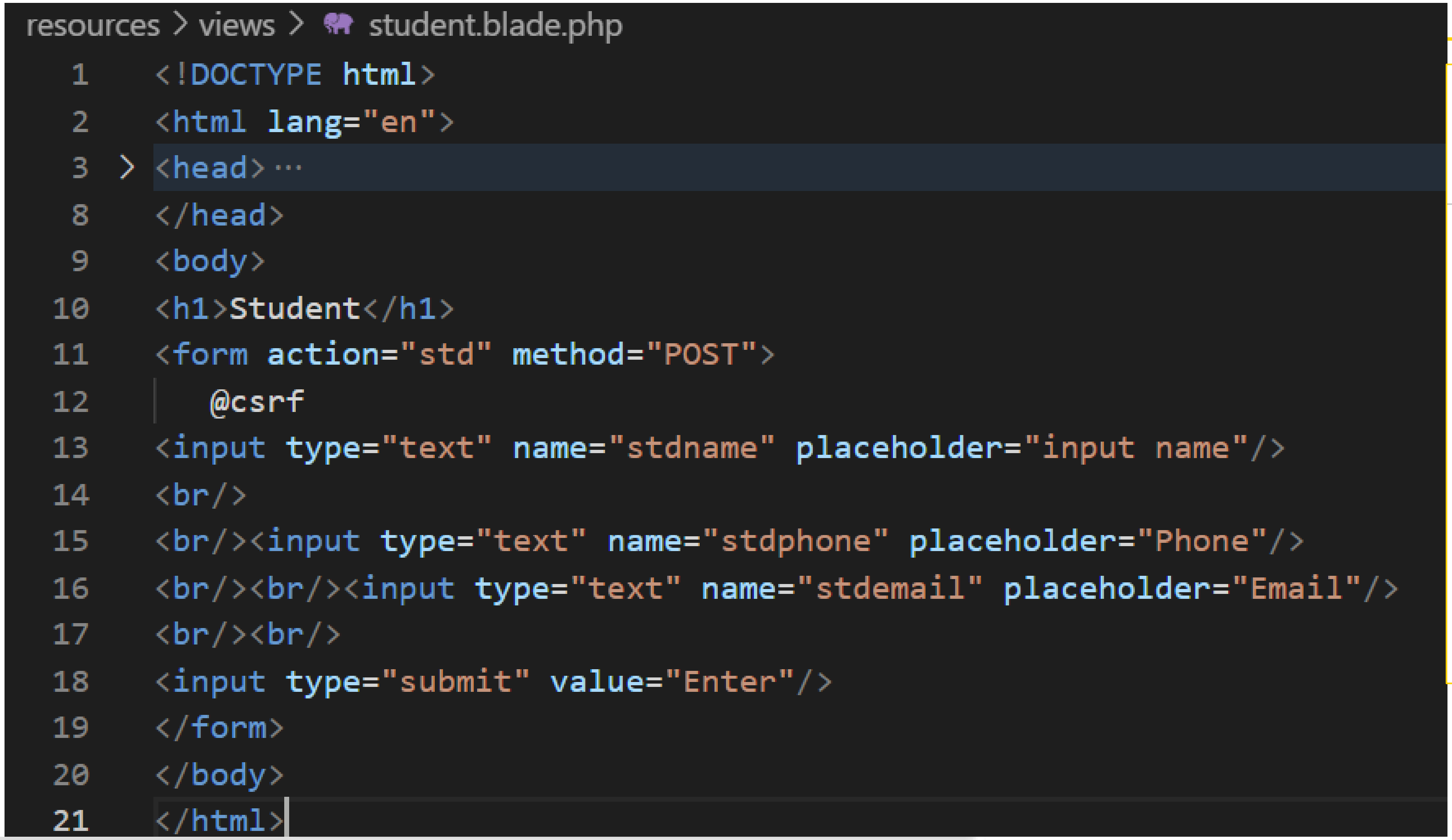
Student Data Request and Response



app > Http > Controllers > StdController.php

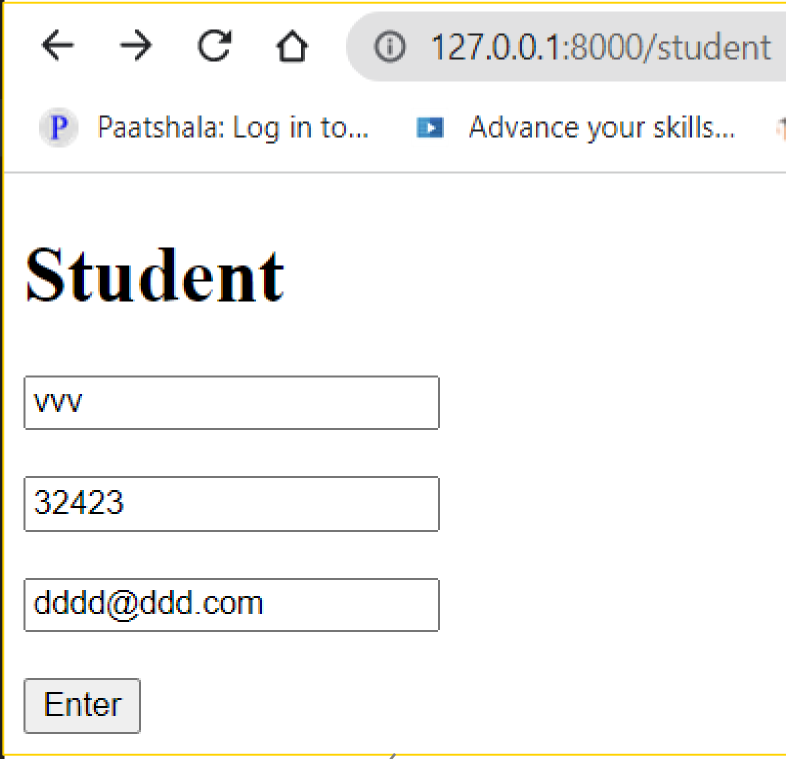
```php
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class StdController extends Controller
{

    function getStdData(Request $req)
    {   //return $req->input();
      //  $input = $req->all();
        $input = $req->getContent();
                echo $input;
    }
}
```

Student Data Request and Response



resources > views > student.blade.php

```
<!DOCTYPE html>
<html lang="en">
<head>···
</head>
<body>
<h1>Student</h1>
<form action="std" method="POST">
    @csrf
<input type="text" name="stdname" placeholder="input name"/>
<br/>
<br/><input type="text" name="stdphone" placeholder="Phone"/>
<br/><br/><input type="text" name="stdemail" placeholder="Email"/>
<br/><br/>
<input type="submit" value="Enter"/>
</form>
</body>
</html>
```



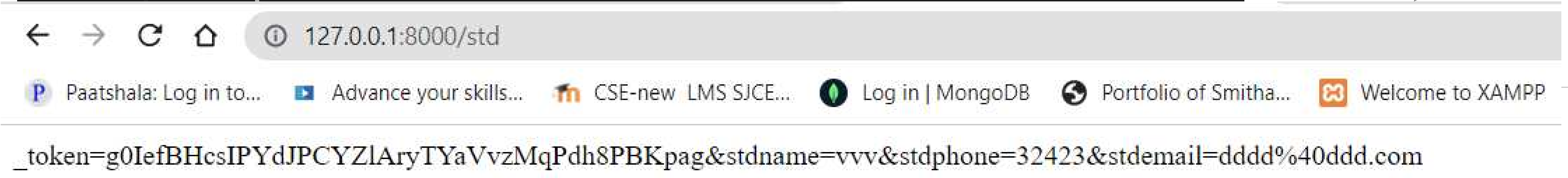
127.0.0.1:8000/student

Student

vvv

32423

dddd@ddd.com



127.0.0.1:8000/std

```
_token=g0IefBHcsIPYdJPCYZlAryTYaVvzMqPdh8PBKpag&stdname=vvv&stdphone=32423&stdemail=dddd%40ddd.com
```