Module 13 Assignment

Part 1: Laravel Installation

To install laravel at first we have to make sure that if PHP is installed or not in our computer. If PHP is installed then we need to install composer. We can install laravel and create projects by following ways:

Download the laravel installer using composer:

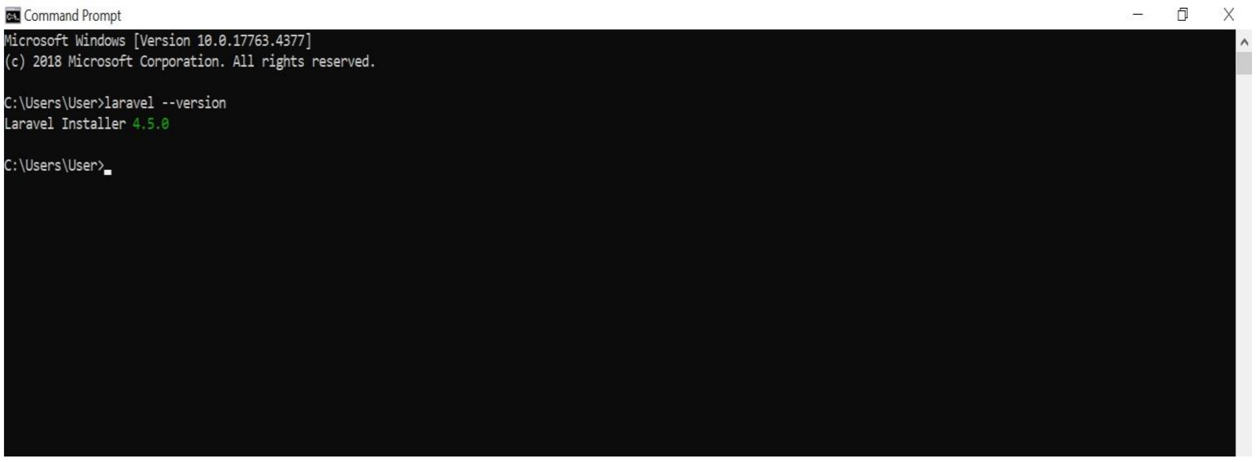
composer global require laravel/installer

Then use laravel new command will create a fresh Laravel installation in the directory that is specified:

laravel new ProjectName

Or we can use following command to create a laravel project:

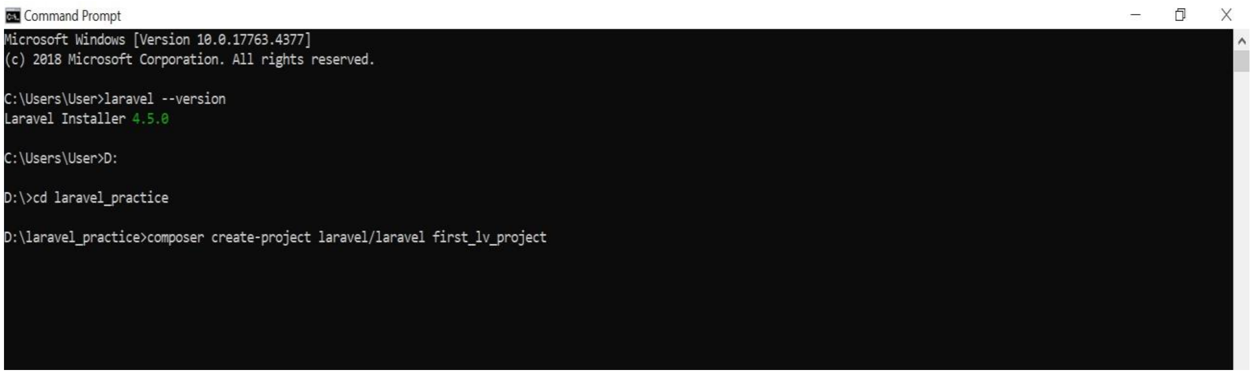
composer create-project laravel/laravel ProjectName



```
Microsoft Windows [Version 10.0.17763.4377]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\User>laravel --version
Laravel Installer 4.5.0

C:\Users\User>
```



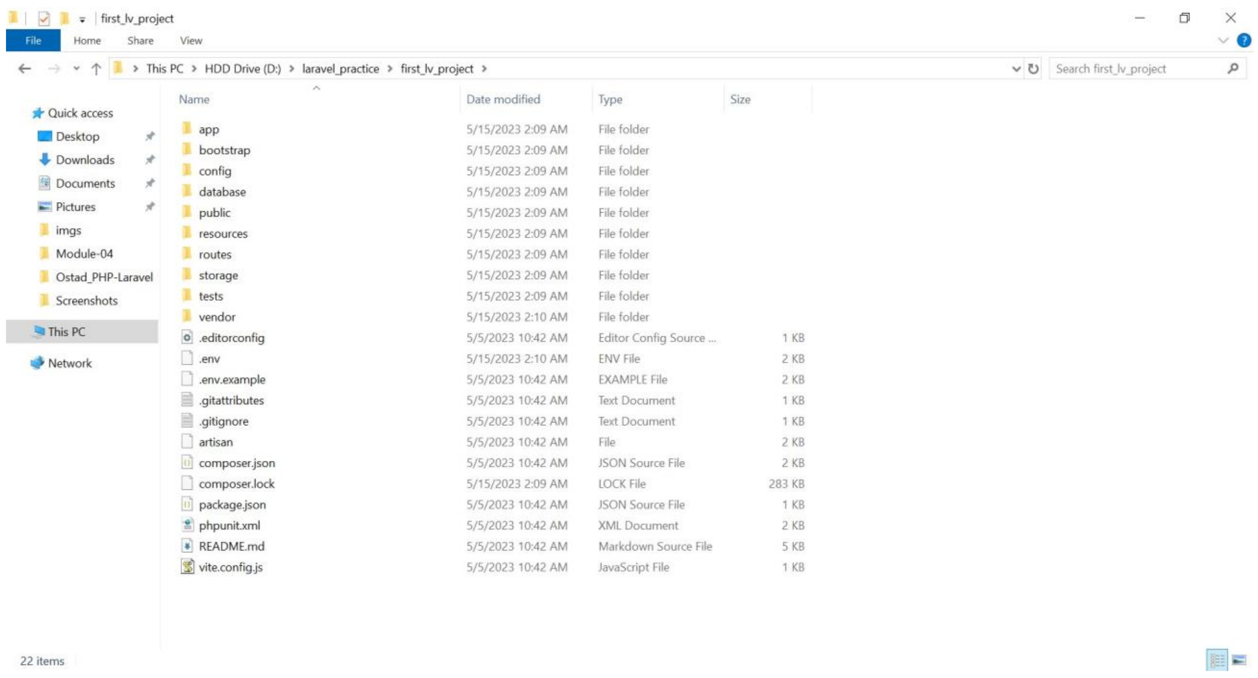
```
Microsoft Windows [Version 10.0.17763.4377]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\User>laravel --version
Laravel Installer 4.5.0

C:\Users\User>D:

D:\>cd laravel_practice

D:\laravel_practice>composer create-project laravel/laravel first_lv_project
```



Name	Date modified	Type	Size
app	5/15/2023 2:09 AM	File folder	
bootstrap	5/15/2023 2:09 AM	File folder	
config	5/15/2023 2:09 AM	File folder	
database	5/15/2023 2:09 AM	File folder	
public	5/15/2023 2:09 AM	File folder	
resources	5/15/2023 2:09 AM	File folder	
routes	5/15/2023 2:09 AM	File folder	
storage	5/15/2023 2:09 AM	File folder	
tests	5/15/2023 2:09 AM	File folder	
vendor	5/15/2023 2:10 AM	File folder	
.editorconfig	5/5/2023 10:42 AM	Editor Config Source ...	1 KB
.env	5/15/2023 2:10 AM	ENV File	2 KB
.env.example	5/5/2023 10:42 AM	EXAMPLE File	2 KB
.gitattributes	5/5/2023 10:42 AM	Text Document	1 KB
.gitignore	5/5/2023 10:42 AM	Text Document	1 KB
artisan	5/5/2023 10:42 AM	File	2 KB
composer.json	5/5/2023 10:42 AM	JSON Source File	2 KB
composer.lock	5/15/2023 2:09 AM	LOCK File	283 KB
package.json	5/5/2023 10:42 AM	JSON Source File	1 KB
phpunit.xml	5/5/2023 10:42 AM	XML Document	2 KB
README.md	5/5/2023 10:42 AM	Markdown Source File	5 KB
vite.config.js	5/5/2023 10:42 AM	JavaScript File	1 KB

22 items

Part 1: Laravel Folder Structure Description

app:

The "app" folder in a Laravel application contains the core files and directories that are responsible for handling the application's logic and business rules. It is one of the most important directories in a Laravel project. Here's a breakdown of the typical contents of the "app" folder:

1. Console: This directory contains all the console commands of your application. Console commands are used to create custom artisan commands that can be executed via the command line.

2. Events: This directory is used to store event classes. Events are used to broadcast messages or trigger actions within your application, allowing you to decouple various components.

3. Exceptions: Here you will find the exception handler class, which handles the exceptions thrown by your application. You can define custom exception classes in this directory to handle specific types of exceptions.

4. Http: The "Http" directory is one of the most important directories within "app". It contains several subdirectories:

   - Controllers: This directory contains your application's controllers. Controllers are responsible for handling incoming requests, processing data, and returning responses.

- Middleware: Middleware classes are used to intercept HTTP requests and perform actions before passing them to the intended route or after the route has been processed.

- Requests: This directory is used to store form request classes. These classes define the rules and validations for incoming HTTP requests.

- Resources: This directory contains the views for your application. It includes subdirectories for "lang" (language files), "views" (Blade templates), and "assets" (CSS, JavaScript, and other static files).

5. Models: This directory typically contains your application's Eloquent models. Eloquent is Laravel's ORM (Object-Relational Mapping) system that provides an easy way to interact with your database tables.

6. Providers: Laravel service providers are responsible for bootstrapping various components of the framework and your application. The "Providers" directory contains service provider classes.

7. Rules: This directory is used to store custom validation rule classes. You can define your own validation rules by creating classes within this directory.

8. Traits: Traits are reusable code snippets that can be included within multiple classes. This directory is used to store trait files.

9. Console.php: This file is the entry point for registering all your console commands. You can define your commands in the "app/Console/Commands" directory and register them in this file.

10. Exceptions.php: This file contains the exception handler configuration.

11. Http.php: This file is used for registering middleware, bindings, and route middleware aliases.

12. Providers.php: In this file, you can register your application's service providers.

Overall, the "app" folder in Laravel organizes essential components of your application, including controllers, models, middleware, views, and more, allowing you to follow the MVC (Model-View-Controller) architectural pattern and build scalable and maintainable web applications.

bootstrap:

In a Laravel project, the "bootstrap" directory plays a crucial role in the application's initialization and bootstrapping process. It contains files and directories that are responsible for setting up the environment, loading essential dependencies, and preparing the application for execution. The "bootstrap" directory is located at the root level of the Laravel project and serves the following purposes:

1. app.php: The "bootstrap/app.php" file is the entry point for the Laravel application. It sets up the basic configuration, registers service providers, and creates the application instance. It loads various components such as the service container, event dispatcher, configuration manager, and other important dependencies.

2. autoload.php: The "bootstrap/autoload.php" file is responsible for autoloading the necessary classes and files used by the application. It includes the Composer's autoloader, which efficiently loads classes and namespaces based on the PSR-4 autoloading standard.

3. cache/: The "bootstrap/cache" directory contains the compiled and cached files generated by Laravel during the application's bootstrapping process. This includes the framework's service container bindings, configuration files, and route cache. Caching these files improves the application's performance by reducing the time required for repetitive operations.

4. config/: The "bootstrap/config" directory contains the configuration files used during the application's bootstrapping phase. These files define various settings and options for Laravel and its components, such as database connections, caching, session management, and more. The configuration files are loaded and merged with the application's runtime configuration.

5. app.php and database.php (optional): The "bootstrap" directory may also contain additional configuration files like "app.php" and "database.php" in some older versions of Laravel. These files were used to specify the application's environment, database connection details, and other settings.

Overall, the "bootstrap" directory is responsible for initializing the Laravel application, loading essential dependencies, and preparing the environment for execution. It ensures that the necessary components and configurations are in place to run the Laravel framework and the application smoothly.

config:

In a Laravel project, the "config" directory is where you can find various configuration files that define the settings and options for different components of your application. The "config" directory is located at the root level of the Laravel project and serves the following purposes:

1. App Configuration: The "config/app.php" file contains the configuration settings for your application. Here, you can define the application name, environment, timezone, locale, encryption key, and other global settings. You can also configure additional features like logging, error handling, and service providers.

2. Database Configuration: The "config/database.php" file is used to configure the database connections for your application. Here, you can specify the database driver (e.g., MySQL, SQLite, PostgreSQL), connection details (host, username, password), and other database-related settings. Laravel supports multiple database connections, and you can define and switch between them as needed.

3. Cache Configuration: The "config/cache.php" file allows you to configure the caching system used by your application. You can define the default cache driver (e.g., file, database, Redis), cache expiration times, cache prefix, and other caching-related options. Caching helps improve the performance of your application by storing frequently accessed data in memory.

4. Session Configuration: The "config/session.php" file is used to configure the session management settings. You can specify the session driver (e.g., file, database, Redis), session lifetime, session encryption, and other session-related options. Sessions allow you to persist user-specific data across multiple requests.

5. Mail Configuration: The "config/mail.php" file contains the configuration settings for sending emails from your application. Here, you can specify the mail driver (e.g., SMTP, sendmail), mail server details, email encryption, and other mail-related options. Laravel provides a unified API for sending emails, making it easy to configure and use different mail services.

6. Queue Configuration: The "config/queue.php" file is used to configure the queueing system in Laravel. You can define the queue driver (e.g., database, Redis, Beanstalkd), connection details, queue worker options, and other queue-related settings. Queues help in offloading time-consuming tasks to be processed asynchronously.

7. Additional Configuration: The "config" directory may also contain other configuration files for specific Laravel components or third-party packages. For example, you might find configuration files for cache drivers, session drivers, logging, filesystems, broadcasting, and more. These files

allow you to customize the behavior of these components according to your application's requirements.

By having a separate "config" directory, Laravel provides a clean and organized way to manage and modify the configuration settings for different parts of your application. It allows you to easily customize and adapt the behavior of Laravel and its components without modifying the core framework files.

database:

In a Laravel project, the "database" directory is where you manage the database-related files and configurations. It contains files and folders that are responsible for defining database migrations, seeders, and handling database connections. The "database" directory is located at the root level of the Laravel project and serves the following purposes:

1. Migrations: The "database/migrations" directory is where you define database migration files. Migrations are a way to version control your database schema. Each migration file represents a set of changes to the database structure, such as creating tables, modifying columns, adding indexes, and more. Laravel provides a convenient syntax for defining migrations, and using the Artisan command-line tool, you can execute and rollback migrations easily.

2. Seeders: The "database/seeders" directory is used for defining seed classes. Seeders allow you to populate your database with initial or dummy data. You can create seed classes to insert data into specific tables or define relationships between records. Using the Artisan command-line tool, you can run seeders to populate your database quickly.

3. Factories: The "database/factories" directory is where you define model factories. Model factories provide a convenient way to generate fake data for testing or database seeding purposes. By defining factories, you can easily create multiple instances of your models with randomized or predefined data.

4. Database Connections: The "database" directory may also contain configuration files for different database connections. By default, the "config/database.php" file specifies the database connections, but you can create additional configuration files to define multiple connections or customize settings for specific environments (e.g., development, production). These configuration files allow you to define the database driver, connection details (host, port, username, password), database name, and other database-specific options.

Overall, the "database" directory in Laravel provides a structured approach to managing your application's database-related tasks. It separates the database migrations, seeders, and factory definitions, allowing you to version control and manage your database schema, populate initial data, and generate fake data easily. The directory also offers flexibility in configuring multiple database connections if needed.

public:

In a Laravel project, the "public" directory serves as the document root for the web server. It contains the publicly accessible files that can be accessed by users through HTTP requests. The "public" directory is located at the root level of the Laravel project and serves the following purposes:

1. Entry Point: The "public/index.php" file is the entry point for all incoming HTTP requests. When a user accesses your Laravel application, the web server routes the request to this file. It initializes the Laravel framework, bootstraps the application, and delegates further processing to the appropriate components.

2. Assets: The "public" directory typically contains various assets required by your application, such as CSS stylesheets, JavaScript files, images, fonts, and other static files. Placing these files in the "public" directory allows them to be directly accessible by the browser. You can organize your assets in subdirectories within the "public" directory for better organization.

3. Front-End Frameworks: If you are using front-end frameworks like Vue.js or React, the "public" directory may contain the entry point files for these frameworks. For example, if you are using Vue.js, you might have a "public/js/app.js" file that serves as the entry point for your Vue.js application.

4. Asset Compilation: During development, you might have separate source files for your CSS and JavaScript assets. These source files are usually stored outside the "public" directory. However, when building your assets for production, Laravel often compiles and minifies them into the "public" directory. This ensures that only the optimized and production-ready assets are served to the users.

5. Publicly Accessible Files: Sometimes, you may need to include certain files in your application that should be directly accessible by users. For example, if you want to provide downloadable files, such as PDFs or media files, you can place them in a subdirectory within the "public" directory. These files can then be accessed using their URLs.

By placing the publicly accessible files in the "public" directory, Laravel provides a security mechanism that ensures only the necessary files are exposed to the outside world. It separates the public assets and entry point files from the rest of the application's code, maintaining a clean project structure and enhancing security.

resources:

In a Laravel project, the "resources" directory is where you store various assets and source files that are used in the frontend development of your application. It is located at the root level of the Laravel project and serves the following purposes:

1. Views: The "resources/views" directory is where you store the Blade templates used to render the HTML output of your application. Blade is Laravel's templating engine that allows you to write concise and expressive views. You can organize your views into subdirectories based on the application's functionality or modules.

2. Localization: The "resources/lang" directory contains language files for localization purposes. Laravel supports multiple languages, and this directory is used to store translation strings for different languages. You can have separate subdirectories for each language, such as "en" for English and "es" for Spanish, and place language files within those directories.

3. Assets: The "resources/assets" (prior to Laravel 8) or "resources" (Laravel 8 onwards) directory is used to store frontend assets such as CSS, JavaScript, images, and other static files. Laravel provides a built-in asset compilation and versioning system called Mix, powered by Laravel Mix, which simplifies the management and building of frontend assets. You can create subdirectories within the "assets" directory to organize your assets based on their types or functionality.

4. SASS/LESS: The "resources/sass" or "resources/less" directory is used to store Sass or Less files, respectively. These preprocessors allow you to write CSS in a more modular and maintainable manner, with features like variables, mixins, and nesting. The Sass or Less files can be compiled into CSS files using Laravel Mix.

5. JavaScript: The "resources/js" directory is where you can store your JavaScript files. Laravel uses the Laravel Mix configuration to compile and bundle these JavaScript files into a single optimized file. You can organize your JavaScript files based on modules or functionality within subdirectories.

6. Components: The "resources/views/components" directory is used to store reusable Blade components. Blade components allow you to encapsulate portions of HTML and logic into

reusable elements. You can create subdirectories within the "components" directory to categorize and organize your components.

7. Layouts and Partials: You can create subdirectories within the "views" directory to store layout files and partial views. Layouts define the overall structure and design of your application's pages, while partial views contain reusable sections of a view, such as headers, footers, sidebars, or widgets.

The "resources" directory in Laravel provides a structured and organized location for managing the frontend assets, views, translations, and other resources required for the presentation layer of your application. It helps separate the frontend code from the backend logic and allows for efficient management and customization of the application's user interface.

routes:

In a Laravel project, the "routes" directory is responsible for defining the routes that map incoming HTTP requests to the corresponding controller actions or closures. The "routes" directory is located at the root level of the Laravel project and serves the following purposes:

1. Web Routes: The "routes/web.php" file contains the routes that handle web requests, typically originating from browsers. Here, you can define routes for various HTTP methods (GET, POST, PUT, DELETE, etc.) and associate them with controller actions or closures. These routes are often used for rendering views, processing form submissions, and performing CRUD operations.

2. API Routes: The "routes/api.php" file contains the routes that handle API requests. These routes are specifically designed for building RESTful APIs. You can define routes for different API endpoints, versioning, and resource controllers. API routes typically return JSON or XML responses and are commonly used for creating, updating, and fetching data via API calls.

3. Console Routes: The "routes/console.php" file contains the routes for defining custom artisan commands. These routes are used to define command-line tasks that can be executed using the Laravel's command-line interface (CLI). You can define custom commands and associate them with closures or command classes to perform specific tasks.

4. Route Caching: Laravel provides an optimization feature called route caching, which can significantly improve the performance of your application by caching the compiled routes. When route caching is enabled, the compiled route files are stored in the "bootstrap/cache" directory, reducing the time required to locate and load the routes on subsequent requests.

By separating routes into different files based on their purposes, Laravel follows the principle of separation of concerns and improves the organization and readability of the codebase. The "routes" directory allows you to define and manage the routing logic in a centralized location, making it easier to maintain and modify the application's routes as your project grows.

storage:

In a Laravel project, the "storage" directory is used for storing various types of data generated or used by the application during its runtime. The "storage" directory is located at the root level of the Laravel project and serves the following purposes:

1. App Storage: The "storage/app" directory is used to store application-specific files that are not accessible to the public. This directory is often used for storing user-generated files, such as uploaded images, documents, or any other type of file that needs to be saved but should not be directly accessible from the web. You can create subdirectories within "storage/app" to organize and manage different types of files.

2. Framework Storage: The "storage/framework" directory contains files generated by Laravel's internal processes. It includes subdirectories such as "cache," "sessions," "testing," and "views." These directories are used by Laravel to store cached data, session files, testing artifacts, and compiled views. The framework automatically manages these directories, creating and deleting files as needed.

3. Logs: The "storage/logs" directory stores the application's log files. Laravel logs various events, errors, and debugging information into these files, making it easier to track and analyze application behavior. Log files can help identify issues, monitor performance, and debug problems in your application.

4. Temporary Files: The "storage/temp" directory (or "storage/framework/temp" in some older versions of Laravel) can be used to store temporary files generated during the application's execution. These files are typically short-lived and can be safely deleted once they are no longer needed. It's a good practice to periodically clean up this directory to avoid accumulating unnecessary temporary files.

5. Publicly Accessible Storage: The "storage/public" directory (or "storage/app/public" in older versions) can be used to store files that need to be publicly accessible through the web server. By default, files placed in this directory are not directly accessible. To make them accessible, you can create symbolic links from the "public/storage" directory to the "storage/app/public" directory using the *“php artisan storage:link”* command.

Overall, the "storage" directory provides a location for managing various types of data generated or used by your Laravel application. It ensures that sensitive files are stored securely, while also providing a convenient place for caching, logging, and temporary file storage. Proper management of the "storage" directory is important for maintaining the integrity, security, and performance of your Laravel application.

tests:

In a Laravel project, the "tests" directory is where you can write and store automated tests for your application. It follows the principles of Test-Driven Development (TDD) and allows you to ensure the correctness and reliability of your code. The "tests" directory is located at the root level of the Laravel project and serves the following purposes:

1. Unit Tests: The "tests/Unit" directory is used to store unit tests. Unit tests focus on testing small, isolated units of code, such as individual methods or functions, to verify that they produce the expected output for a given set of inputs. Unit tests are typically written for models, helper functions, utility classes, and other self-contained units.

2. Feature Tests: The "tests/Feature" directory is used to store feature tests. Feature tests are higher-level tests that focus on testing the functionality and behavior of a feature or a specific scenario of your application. They simulate user interactions and HTTP requests to test the application's routes, controllers, and responses.

3. Browser Tests: The "tests/Browser" directory is used to store browser tests. Browser tests, also known as end-to-end tests or acceptance tests, simulate user interactions in a real browser environment. These tests are written using Laravel Dusk, a browser automation and testing tool provided by Laravel. Browser tests allow you to test the application's frontend components, such as form submissions, AJAX requests, and user interface interactions.

4. PHPUnit Configuration: The "tests" directory may contain a "phpunit.xml" or "phpunit.xml.dist" file, which is the configuration file for PHPUnit, the testing framework used by Laravel. This file specifies various settings and options for running tests, such as test directories, test suites, code coverage, and more.

5. Custom Test Classes: Besides the default "Unit," "Feature," and "Browser" directories, you can create additional directories and custom test classes based on your application's needs. For example, you may create a directory for API tests, integration tests, or specific module-related tests.

Laravel provides a robust testing framework with various helper methods, assertions, and testing utilities to simplify the process of writing and executing tests. Running tests can be done using the artisan command-line interface or through Continuous Integration (CI) systems. By writing tests, you can detect and prevent regressions, ensure the stability of your application, and confidently make changes or additions to your codebase.

vendor:

In a Laravel project, the "vendor" directory contains all the third-party dependencies and libraries that your application relies on. The "vendor" directory is created and managed by Composer, a dependency management tool used by Laravel.

When you start a new Laravel project or add new packages to an existing project, Composer automatically downloads and installs the required dependencies into the "vendor" directory. The purpose of the "vendor" directory is as follows:

1. Third-Party Packages: The "vendor" directory houses all the third-party packages and libraries that your Laravel project depends on. This includes popular packages like Laravel's own dependencies, as well as additional packages that you have added to enhance your application's functionality. Each package resides in its own subdirectory within the "vendor" directory.

2. Autoloading: Composer generates an autoloading mechanism based on the installed packages in the "vendor" directory. This allows your application to easily access classes and functions from the installed packages without the need for manual inclusion. Composer's autoloader ensures that the necessary files are automatically included when they are referenced in your code.

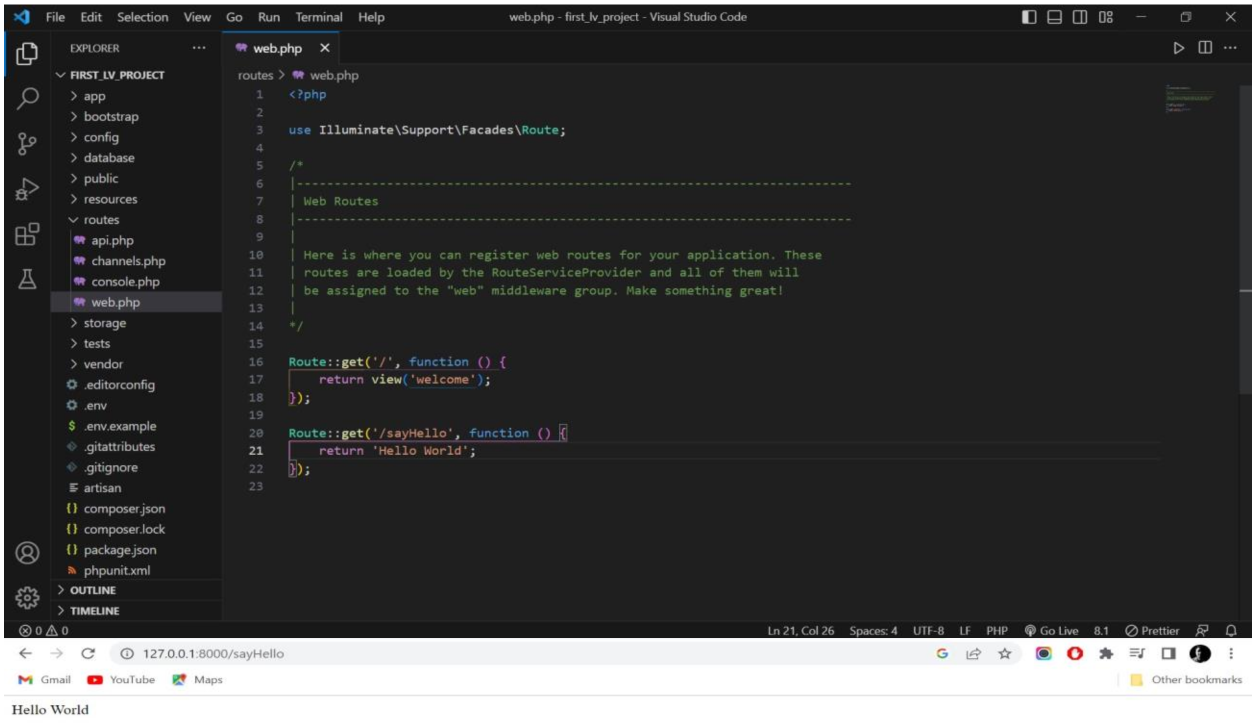
3. Update and Dependency Management: Composer provides commands to update, install, and manage your project's dependencies. It maintains a "composer.json" file in the root of your Laravel project that lists all the required packages and their versions. By running Composer commands, you can update packages to newer versions, install new packages, and resolve dependencies automatically.

4. Vendor Libraries and Assets: Some packages may include additional assets such as CSS, JavaScript, images, or configuration files. These files are usually located within the package's subdirectory in the "vendor" directory. Depending on the package, you may need to publish or configure these assets to make them accessible within your application.

It's important to note that the "vendor" directory should not be modified manually. It is intended to be managed by Composer, and any modifications or updates should be done through Composer commands. This ensures consistency and reproducibility across different environments and allows for seamless collaboration when working with other developers on the project.

Overall, the "vendor" directory is a crucial part of a Laravel project as it manages and organizes the third-party dependencies and libraries required by your application, making it easier to integrate and leverage the power of external packages within your Laravel application.

Below the screenshots of creating new route in laravel project to display “Hello World” are shown:



```php
<?php

use Illuminate\Support\Facades\Route;

/*
|--------------------------------------------------------------------------
| Web Routes
|--------------------------------------------------------------------------
|
| Here is where you can register web routes for your application. These
| routes are loaded by the RouteServiceProvider and all of them will
| be assigned to the "web" middleware group. Make something great!
|
*/

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

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

```

```
Hello World
```