



Laravel

Session Outline

€Introduction To Laravel Framework

€Key Features of Laravel

€Laravel Installation

€MVC Architecture

€Laravel Application Structure

Introduction to Laravel

- Laravel is a popular open-source PHP web framework designed to make **web development easier** and **more efficient**.
- It follows the Model-View-Controller (**MVC**) architectural pattern and provides tools and features to simplify tasks such as **routing, authentication, sessions, caching, and database management**.
- Laravel's goal is to **improve the development process** by offering expressive, elegant syntax while allowing developers to focus on the **core logic** of their applications.

Key Features of Laravel

1. MVC Architecture

Laravel follows the **Model-View-Controller (MVC)** architectural pattern, which separates the application's logic from its user interface.

- **Model** handles the data and business logic.
- **View** is responsible for the user interface.
- **Controller** acts as a bridge, managing communication between the Model and View.

Benefit: This separation makes it easier to organize code, improving readability and maintainability.

Cont. ...

2. Eloquent ORM

Eloquent is Laravel's built-in **Object-Relational Mapping (ORM)** tool, which allows developers to interact with the database using PHP syntax rather than raw SQL.

- It follows an ActiveRecord pattern, where each model corresponds to a database table, and each instance of the model represents a row in the table.

Benefit: Eloquent simplifies database operations like CRUD (Create, Read, Update, Delete) through easy-to-use methods.

Cont. ...

3. Blade Templating Engine

Laravel provides **Blade**, a lightweight yet powerful templating engine for designing dynamic web pages.

- It allows embedding PHP code within views while offering control structures like loops and conditionals.
- Blade also enables template inheritance, which helps reuse layout components (e.g., headers, footers) efficiently.

Benefit: Blade simplifies the creation of views with minimal PHP code and faster execution.

Cont. ...

4. Artisan CLI

Artisan is Laravel's command-line interface that automates many repetitive tasks, such as:

- Running database migrations
- Creating controllers, models, or other components
- Running unit tests
- Scheduling tasks

Benefit: Artisan helps streamline development workflows, saving time and reducing manual errors.

Cont. ...

5. Routing System

Laravel offers an expressive and simple-to-use **routing** system to manage web requests. Developers can easily define routes for their application, grouping them or applying middleware.

- **Route caching** improves the performance of applications with a large number of routes.

Benefit: The routing system is intuitive and supports complex functionalities like route grouping, naming, and middleware integration.

Cont. ...

6. Middleware

Laravel's **middleware** provides a way to filter HTTP requests before they reach the controller. For example, authentication middleware can be used to ensure that only authenticated users can access certain parts of the application.

Benefit: Middleware adds a layer of security and helps manage requests efficiently by separating cross-cutting concerns like logging and authentication.

Cont. ...

7. Database Migrations and Schema Builder

Laravel provides **migrations** to version-control the database schema. With migrations, you can create or modify database tables programmatically, keeping schema changes in sync across environments.

- The **Schema Builder** simplifies defining and managing database tables and columns through code.

Benefit: Migrations make it easy to track and roll back database changes, facilitating teamwork in multi-developer projects.

Cont. ...

8. Authentication and Authorization

Laravel includes **built-in authentication** mechanisms, including user login, registration, and password recovery. It also supports **authorization** with roles and permissions.

- **Gates and Policies** are used to define and enforce access control logic within the application.

Benefit: Laravel provides a quick and secure way to implement user management and access control without building it from scratch.

Cont. ...

9. Task Scheduling

Laravel provides a simple, fluent API for scheduling tasks using the **Artisan command scheduler**. You can schedule tasks such as sending out notifications or pruning old records.

Benefit: Task scheduling is straightforward, eliminating the need for cron job management in the server.

10. RESTful APIs

Laravel supports the building of **RESTful APIs** quickly and efficiently. The framework offers features like resource routing and data formatting, making it easier to create scalable APIs.

- Built-in features like API rate limiting and response formatting enhance security and usability.

Benefit: Laravel simplifies API development with minimal code, reducing the complexity of creating backend services.

Laravel - Installation

Installing Laravel 11: A Step-by-Step Guide

#webdev #beginners #programming #laravel

Laravel 11 is a powerful PHP framework that helps developers build robust and scalable web applications. This guide will walk you through the installation process and outline the dependencies required to get your Laravel 11 application up and running.

Cont. ...

Prerequisites

Before you install Laravel 11, ensure you have the following prerequisites installed on your machine:

1. **PHP:** Laravel 11 requires PHP 8.1 or higher.
2. **Composer:** Laravel uses Composer to manage its dependencies.
3. **Web Server:** Apache or Nginx is recommended.
4. **Database:** MySQL, PostgreSQL, SQLite, or SQL Server.

Step 1: Install PHP

Make sure you have PHP 8.1 or higher installed. You can download the latest version of PHP from the [official PHP website](#).

Verify the installation by running:

```
php -v
```

Cont. ...

Step 2: Install Composer

Composer is a dependency manager for PHP. Download and install Composer from the [official Composer website](#).

Verify the installation by running:

```
composer -v
```

Step 3: Install Laravel 11

With PHP and Composer installed, you can now install Laravel 11. Open your terminal and run the following command:

```
composer create-project --prefer-dist laravel/laravel laravel11-app "11.*"
```

This command will create a new Laravel 11 project in a directory named `laravel11-app`.

Cont. ...

Step 4: Configure Environment

Navigate to your project directory:

```
cd laravel11-app
```

Copy the `.env.example` file to `.env`:

```
cp .env.example .env
```

Generate a new application key:

```
php artisan key:generate
```

Update your `.env` file with your database credentials and other necessary configurations.

Cont. ...

Step 5: Set Up a Web Server

Using Artisan Serve (Development Only)

For development purposes, you can use Laravel's built-in server:

```
php artisan serve
```

Visit `http://localhost:8000` in your browser to see your Laravel application.

Output

The screenshot displays the Laravel website's output. At the top, a dark navigation bar contains a back arrow, a forward arrow, a refresh icon, a shield icon, a document icon, the URL "127.0.0.1:8000", a star icon, a heart icon, a person icon, a share icon, and a menu icon. Below the navigation bar, the main content area features a large red-to-black gradient background. In the center, the Laravel logo is displayed. On the left side, there is a dark sidebar with the Laravel logo and a list of links. The main content area is divided into four sections: 1. Documentation: A dark box with the Laravel logo, a search bar, and a list of links. 2. Laracasts: A dark box with a video icon, the title "Laracasts", and a description: "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." 3. Laravel News: A dark box with a document icon, the title "Laravel News", and a description: "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." 4. Vibrant Ecosystem: A dark box with a globe icon, the title "Vibrant Ecosystem", and a description: "Laravel's robust library of first-party tools and libraries, such as Forge, Envoy, and Horizon, make it easy to build and deploy your applications." At the bottom left, the date "11/14/2024" is displayed. At the bottom center, the text "Introduction to Laravel Framework" is visible. At the bottom right, the page number "18" is shown.

127.0.0.1:8000

Laravel

Q Search

VERSION

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.

Vibrant Ecosystem

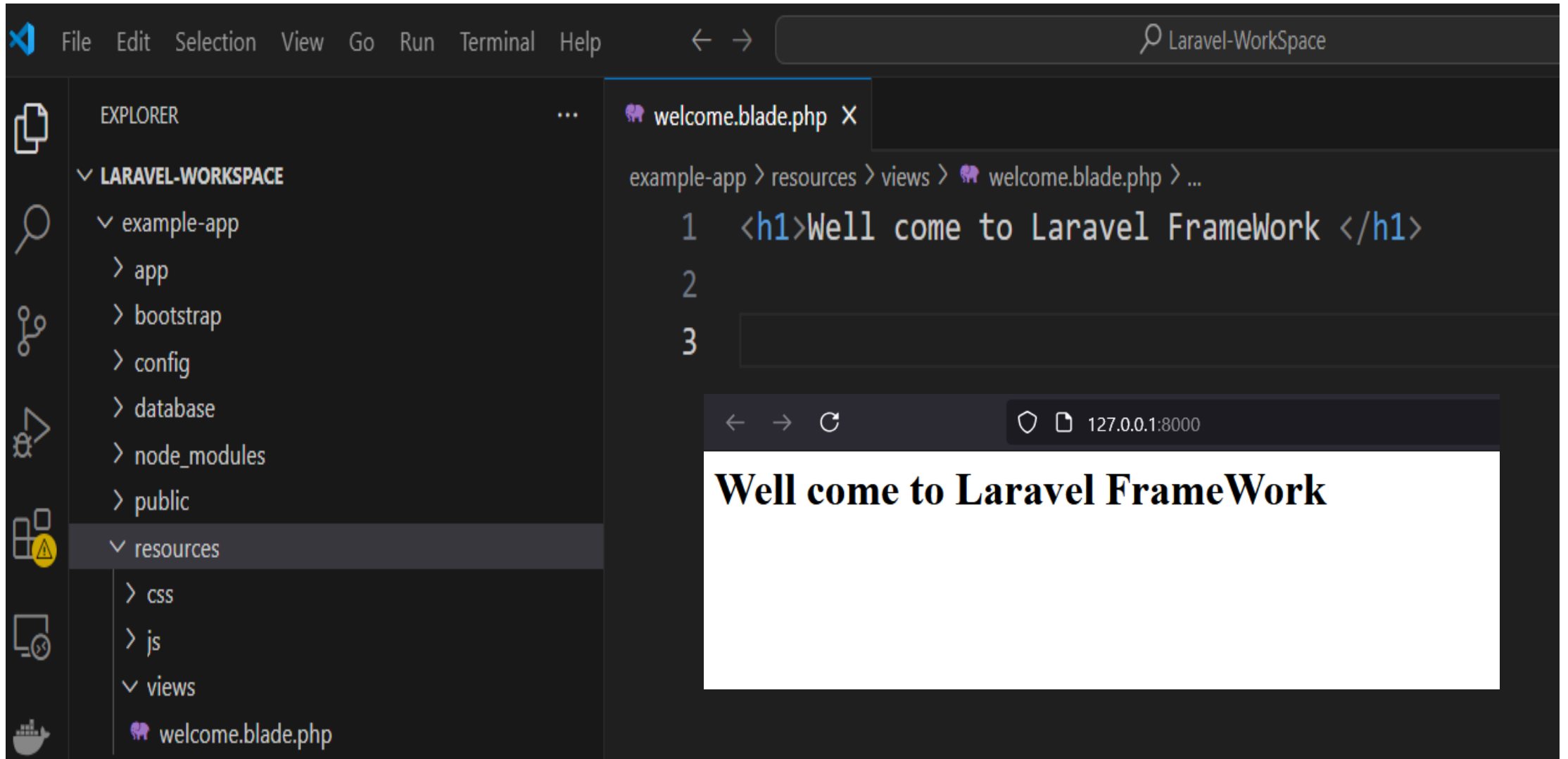
11/14/2024

Documentation

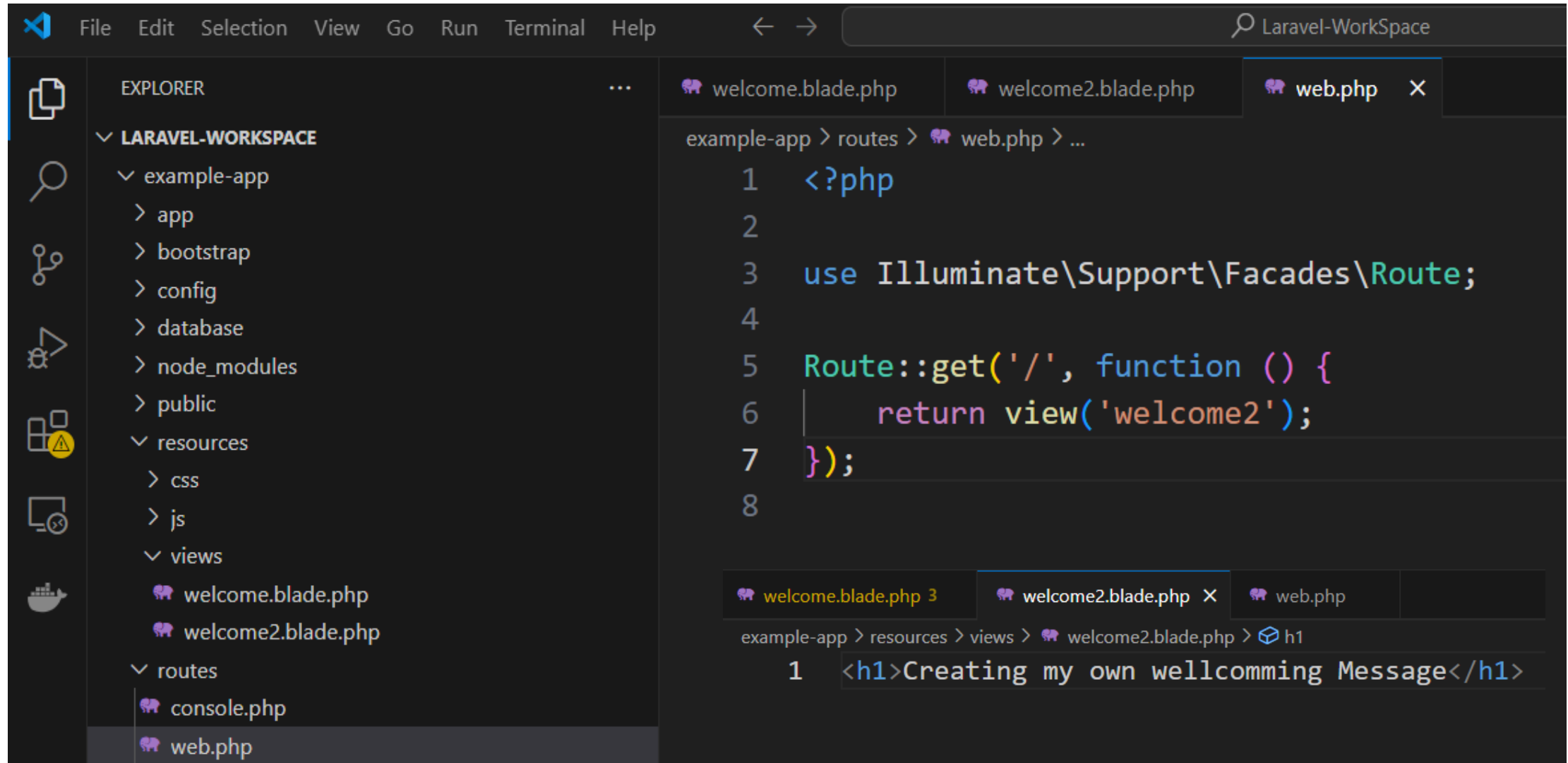
Introduction to Laravel Framework

18

How to Change View



Creating our Owen File



The screenshot shows the Visual Studio Code interface with the Laravel-Workspace project open. The Explorer sidebar on the left displays the project structure, including the 'resources' directory and its subdirectories 'css', 'js', and 'views'. The 'views' directory is expanded, showing 'welcome.blade.php', 'welcome2.blade.php', and 'web.php'. The 'web.php' file is selected and open in the editor. The editor shows the code for 'web.php' in the top pane and 'welcome2.blade.php' in the bottom pane. The top pane contains PHP code for a route that returns a view. The bottom pane contains Blade template code for an h1 heading.

```
File Edit Selection View Go Run Terminal Help
```

EXPLORER

LARAVEL-WORKSPACE

- example-app
 - app
 - bootstrap
 - config
 - database
 - node_modules
 - public
 - resources
 - css
 - js
 - views
 - welcome.blade.php
 - welcome2.blade.php
 - web.php

welcome.blade.php welcome2.blade.php web.php

example-app > routes > web.php > ...

```
1 <?php
2
3 use Illuminate\Support\Facades\Route;
4
5 Route::get('/', function () {
6     return view('welcome2');
7 });
8
```

welcome.blade.php 3 welcome2.blade.php web.php

example-app > resources > views > welcome2.blade.php > h1

```
1 <h1>Creating my own wellcomming Message</h1>
```

Cont. ...

Step 6: Install Dependencies

Laravel 11 comes with several dependencies pre-installed. However, you may need to install additional packages depending on your application's requirements. Here are some common dependencies:

- **Database:** Install the database driver for your chosen database (e.g., `pdo_mysql` for MySQL).
- **Cache:** For caching, you might want to install a Redis or Memcached driver.
- **Queue:** For job queues, you might use Redis, Beanstalkd, or Amazon SQS.

You can install additional packages using Composer. For example, to install the Redis driver, run:

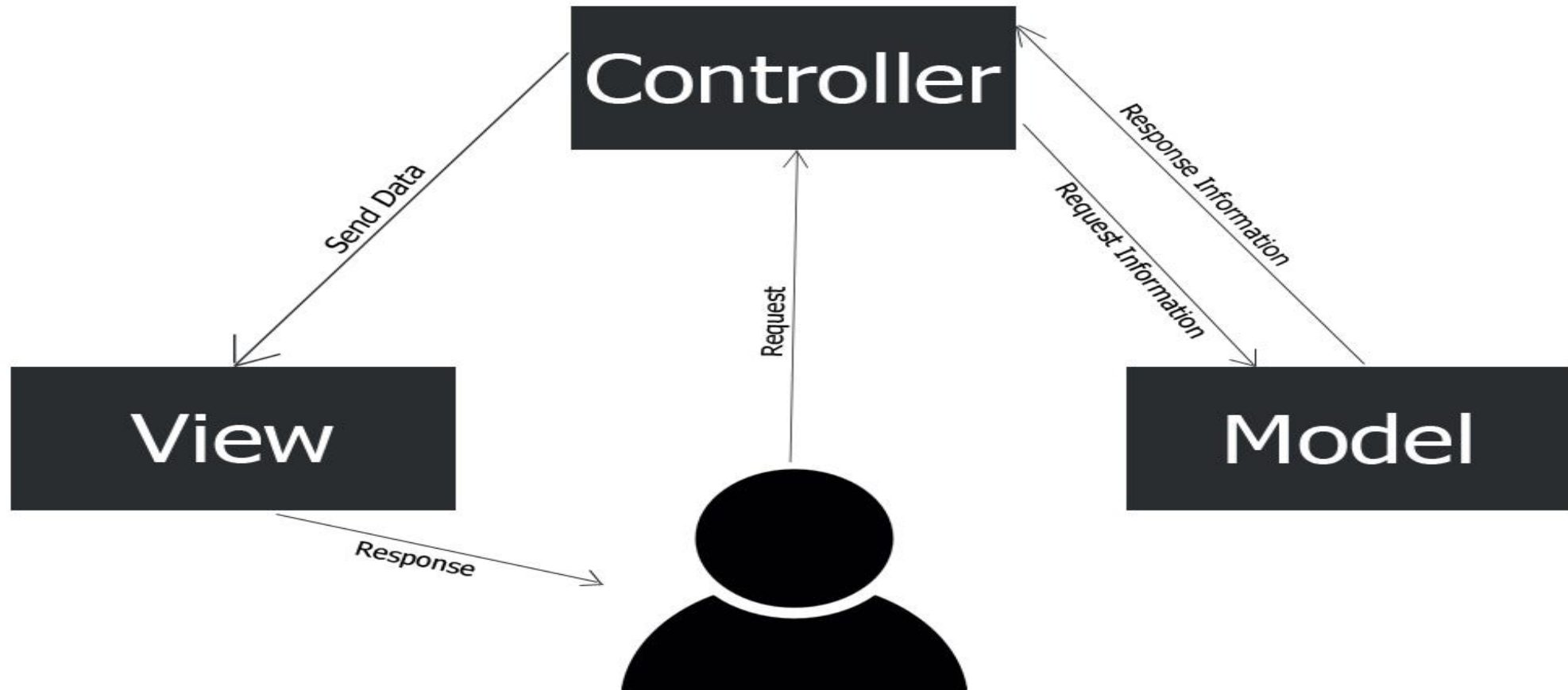
```
composer require predis/predis
```

MVC (Model-View-Controller)

- MVC (Model-View-Controller) is a **design pattern** used in software development to separate an application into **three interconnected** components: **Model**, **View**, and **Controller**.
- This separation helps manage the complexity of large applications, improves maintainability, and promotes the **reuse of code**.
- It is widely adopted in web development frameworks like **Laravel**, **Ruby on Rails**, and **Django**.

MVC Architecture

Model-View-Controller



Model


1. Model

- **Role:** The **Model** represents the data and the business logic of the application. It is responsible for managing the application's data, interacting with the database, defining the relationships between data, and handling validation or rules.
- **Purpose:** Models retrieve data from databases, perform any necessary computations or logic, and return the processed data to the Controller.
- **Example:** In a blogging system, the **Post** model would interact with the database to retrieve all posts, create new posts, update existing posts, or delete posts.

Cont. ...

Example of a Model in PHP (without any framework):

php

 Copy code

```
<?php


class Post {
    private $db;

    // Constructor connects to the database
    public function __construct($dbConnection) {
        $this->db = $dbConnection;
    }
}
```

Cont. ...

```
// Method to retrieve all posts from the database
public function getAllPosts() {
    $query = "SELECT * FROM posts";
    $result = $this->db->query($query);

    return $result->fetchAll(PDO::FETCH_OBJ); // Returns the data as an array of objects
}
}
```

 Copy code

In this example:

- The `Post` class represents the model.
- The `getAllPosts()` method retrieves all posts from the database.

View

2. View

- **Role:** The **View** is responsible for presenting data to the user. It defines how the user interface will look and what data will be displayed. Views are often HTML files, but they can also be other formats like JSON or XML in the context of APIs.
- **Purpose:** The View retrieves the data sent from the Controller and presents it to the user in a structured and readable format.
- **Example:** In a blog system, a `post.blade.php` file in Laravel would display the title, content, and author of a blog post.

Cont. ...

```
C:\Users\yitayew\Desktop\form.html - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

form.html x
1 <!-- post-list.php -->
2 <!DOCTYPE html>
3 <html lang="en">
4 <head>
5     <meta charset="UTF-8">
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7     <title>Blog Posts</title>
8 </head>
9 <body>
10    <h1>Blog Posts</h1>
11    <?php if (!empty($posts)) : ?>
12        <ul>
13            <?php foreach ($posts as $post) : ?>
14                <li>
15                    <h2><?php echo $post->title; ?></h2>
16                    <p><?php echo $post->content; ?></p>
17                </li>
18            <?php endforeach; ?>
19        </ul>
20    <?php else : ?>
21        <p>No posts available.</p>
22    <?php endif; ?>
23 </body>
24 </html>
```

In this example:

- The **View** (`post-list.php`) is an HTML file with embedded PHP.
- It expects a `$posts` array from the controller and renders each blog post.


Controller

3. Controller

- **Role:** The **Controller** acts as an intermediary between the Model and the View. It receives requests from the user (typically through HTTP), processes them (by interacting with the Model), and returns the appropriate View as a response. The Controller contains the application logic and coordinates how data flows between the Model and the View.
- **Purpose:** The Controller gathers input, processes it, and decides which view to display or what action to take.
- **Example:** In a blog system, the `PostController` would handle requests to view a list of posts, create a new post, or edit an existing one.

Cont. ...

<?php

 Copy code

```
class PostController {
    private $postModel;

    // Constructor accepts a model instance
    public function __construct($postModel) {
        $this->postModel = $postModel;
    }

    // Method to handle displaying all posts
    public function index() {
        // Fetch all posts from the model
        $posts = $this->postModel->getAllPosts();

        // Include the view file and pass the posts data
        include 'views/post-list.php';
    }
}
```

In this example:

- The `PostController` interacts with the `Post` model.
- The `index()` method retrieves all posts and then includes the `post-list.php` view, passing the `$posts` data to it.

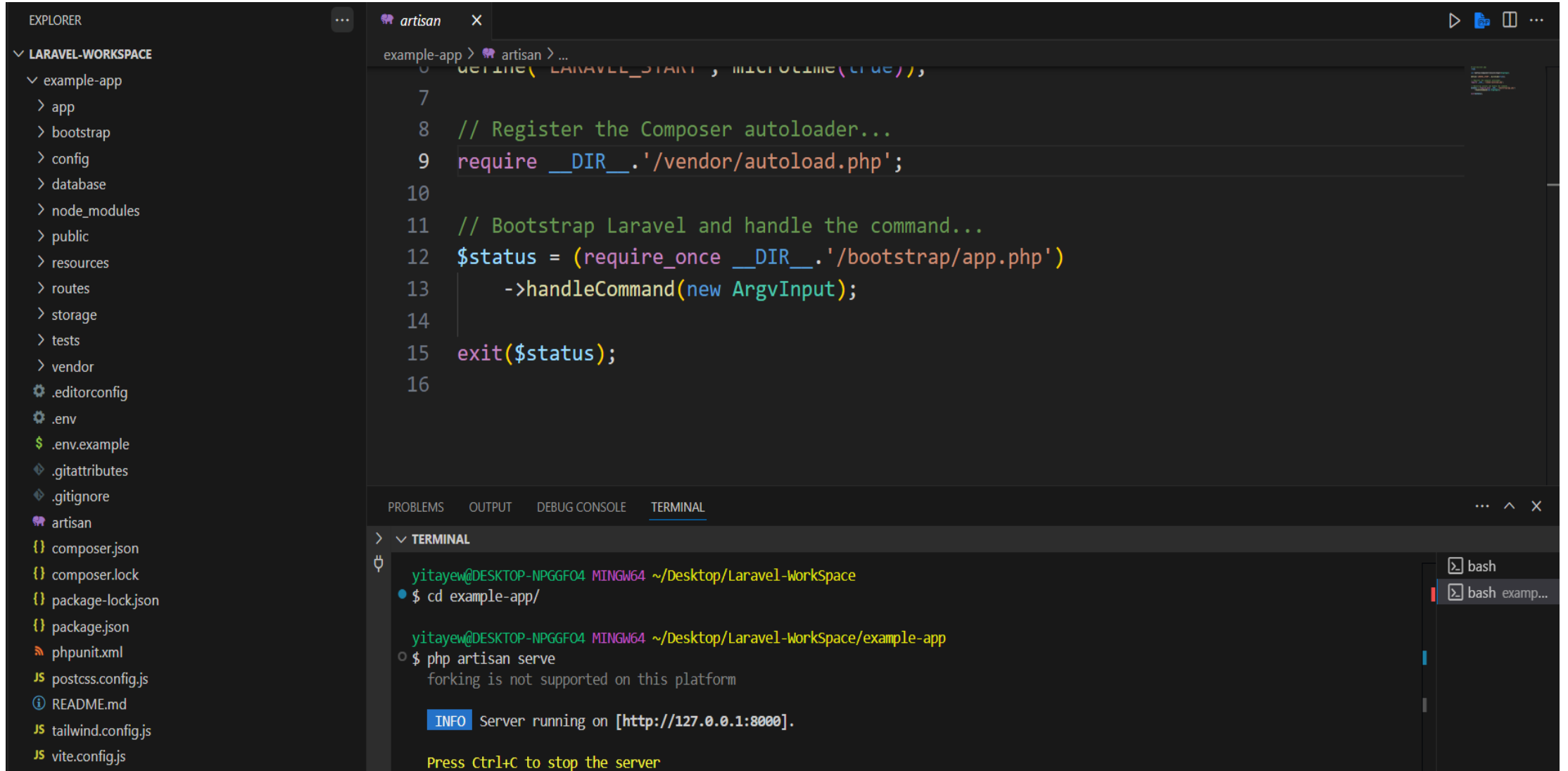
Benefits of MVC

- ❖ **Separation of Concerns:** Each component has a clear responsibility, making the application easier to maintain and update.
- ❖ **Reusability:** Models, Views, and Controllers can be reused across different parts of the application.
- ❖ **Scalability:** MVC helps in scaling an application since components are loosely coupled and can be modified independently.
- ❖ **Testability:** The separation allows for easier testing of individual components like models or controllers.

Laravel - Application Structure

- The application structure in Laravel is basically the **structure of folders, sub-folders and files** included in a project. Once we create a project in Laravel, we get an overview of the application structure as shown in the image here.
- The snapshot shown here refers to the root folder of Laravel namely **example-app**. It includes various sub-folders and files. The analysis of folders and files, along with their functional aspects is given below

Cont. ...



The screenshot shows the Visual Studio Code interface with a Laravel workspace. The Explorer sidebar on the left lists the project structure, including directories like `app`, `bootstrap`, `config`, `database`, `node_modules`, `public`, `resources`, `routes`, `storage`, `tests`, and `vendor`, as well as configuration files like `.editorconfig`, `.env`, `.env.example`, `.gitattributes`, `.gitignore`, `artisan`, `composer.json`, `composer.lock`, `package-lock.json`, `package.json`, `phpunit.xml`, `postcss.config.js`, `README.md`, `tailwind.config.js`, and `vite.config.js`.

The main editor displays the `artisan` file with the following PHP code:

```
1  DEFINE('LARAVEL_START', microtime(true));
2
3  // Register the Composer autoloader...
4  require __DIR__.'/vendor/autoload.php';
5
6  // Bootstrap Laravel and handle the command...
7
8  $status = (require_once __DIR__.'/bootstrap/app.php')
9      ->handleCommand(new ArgvInput);
10
11 exit($status);
```

The bottom panel shows the TERMINAL output:

```
> yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-WorkSpace
• $ cd example-app/

yitayew@DESKTOP-NPGGF04 MINGW64 ~/Desktop/Laravel-WorkSpace/example-app
o $ php artisan serve
  forking is not supported on this platform

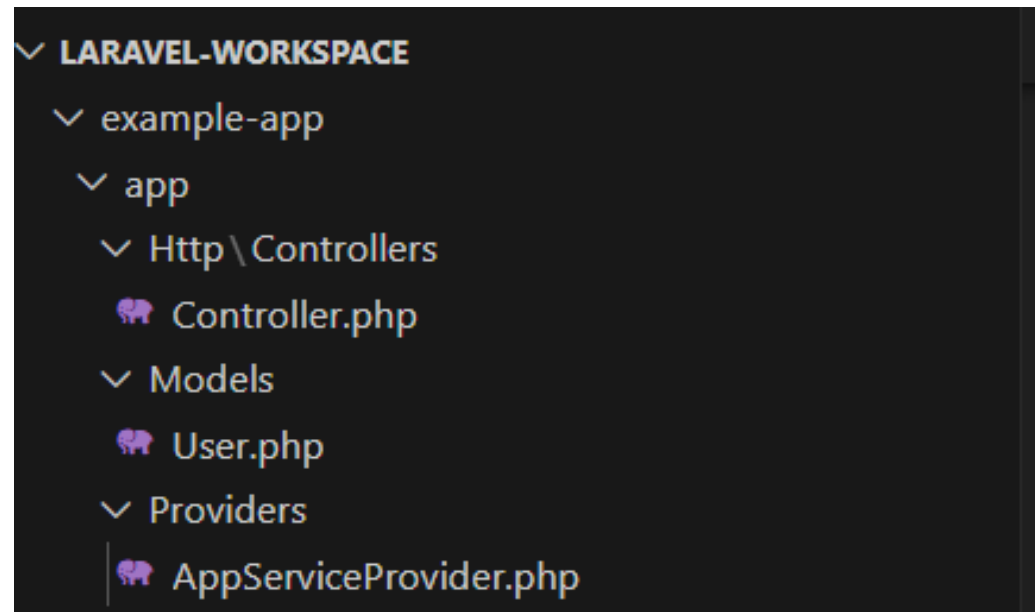
  INFO Server running on [http://127.0.0.1:8000].

Press Ctrl+C to stop the server
```

On the right side of the terminal, there are two tabs: `bash` and `bash example...`.

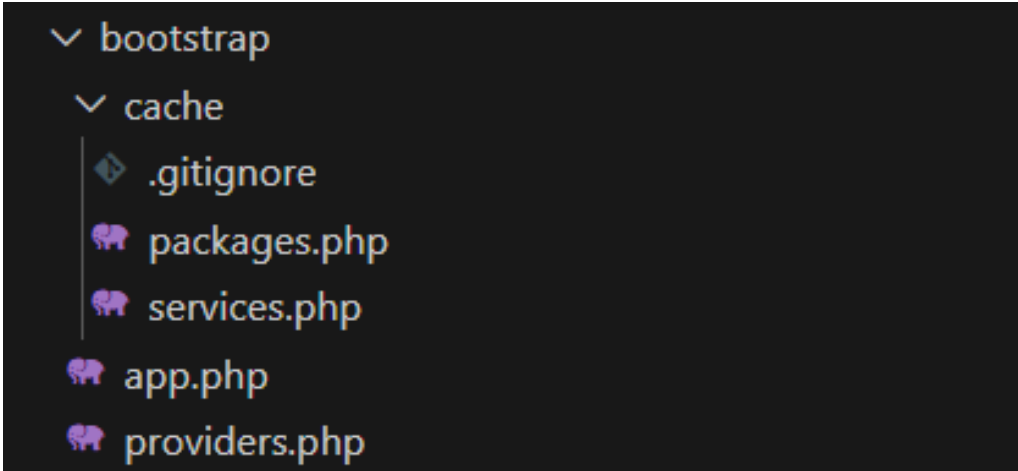
App

- The app directory contains the **core code of your application**. We'll explore this directory in more detail soon; however, almost all of the **classes** in your application will be in this directory.



The Bootstrap Directory

- The bootstrap directory contains the app.php file which **bootstraps** the **framework**. This directory also houses a **cache directory** which contains framework generated files for performance optimization such as the route and services cache files.



```

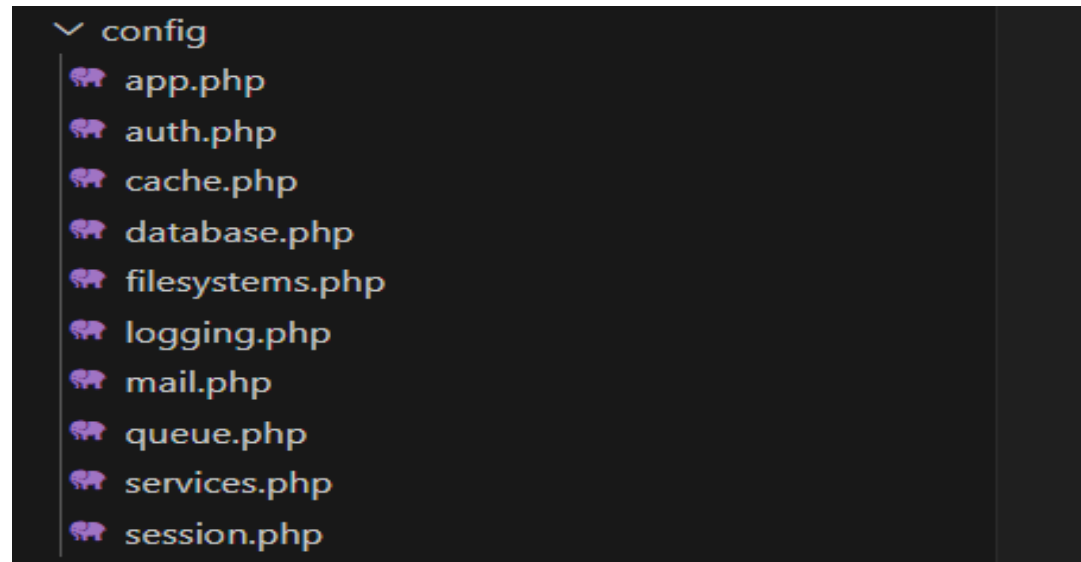
✓ bootstrap
  ✓ cache
    .gitignore
    packages.php
    services.php
  app.php
  providers.php

```

A screenshot of a file explorer interface with a dark background. It shows a tree view of the 'bootstrap' directory. Under 'bootstrap', there is a 'cache' subdirectory and several files. The 'cache' directory is expanded, showing a '.gitignore' file and two PHP files: 'packages.php' and 'services.php'. Below the 'cache' directory, there are two more PHP files: 'app.php' and 'providers.php'. Each file is preceded by a small purple elephant icon, which is the Laravel logo.

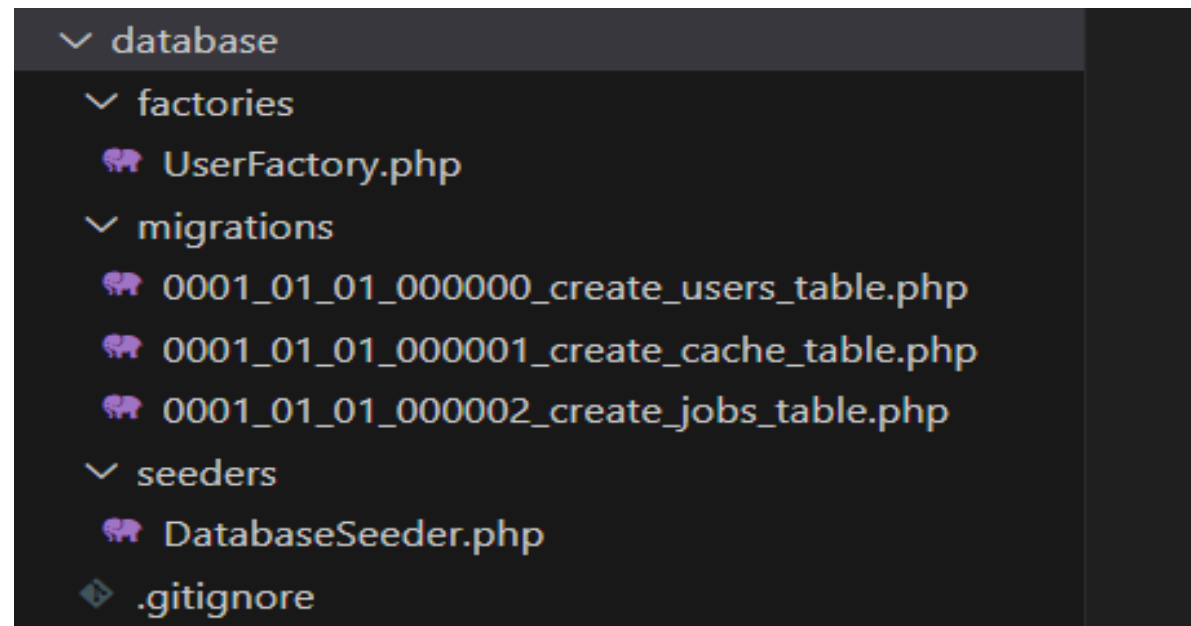
The Config Directory

- The config directory, as the name implies, contains all of your application's configuration files. It's a great idea to read through all of these files and familiarize yourself with all of the options available to you.



The Database Directory

- The database directory contains your **database migrations**, **model factories**, and **seeds**. If you wish, you may also use this directory to hold an SQLite database.



The Public Directory

- 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

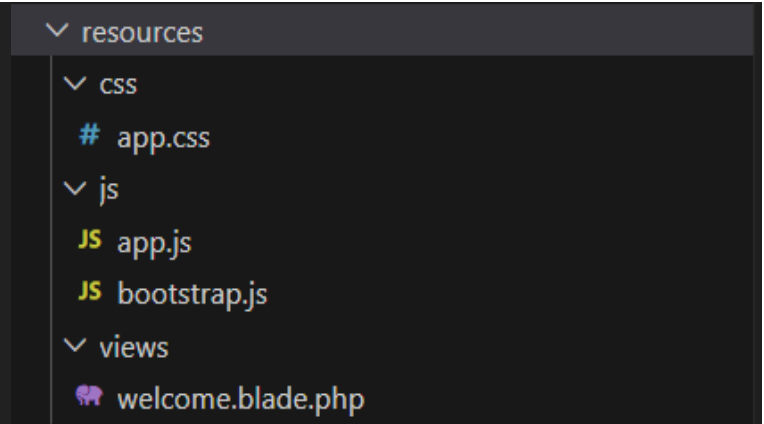
6. resources/

This folder contains all of your application's front-end resources.

- Subdirectories:

- `views/` : Contains Blade templates (Laravel's templating engine).
- `lang/` : Contains language files for localization.
- `js/` , `css/` , `sass/` : Houses frontend assets like JavaScript, CSS, and SCSS files.

Purpose: Manages front-end templates, assets, and localization files.



The Routes Directory

- The routes directory contains all of the route definitions for your application. By default, two route files are included with Laravel: **web.php** and **console.php**.
- The **web.php** file contains **routes** that Laravel places in the web middleware group, which provides **session state**, **CSRF protection**, and **cookie encryption**. If your application does not offer a stateless, RESTful API then all your routes will most likely be defined in the web.php file.

Cont. ...

- The **console.php** file is where you may define all of your **closure** based **console commands**. Each closure is bound to a command instance allowing a simple approach to interacting with each command's IO methods.
- Even though this file does not define HTTP routes, it defines console based entry points (routes) into your application. You may also schedule tasks in the console.php file.
- Optionally, you may install additional route files for API routes (api.php) and broadcasting channels (channels.php), via the install:api and install:broadcasting Artisan commands.

Cont. ...

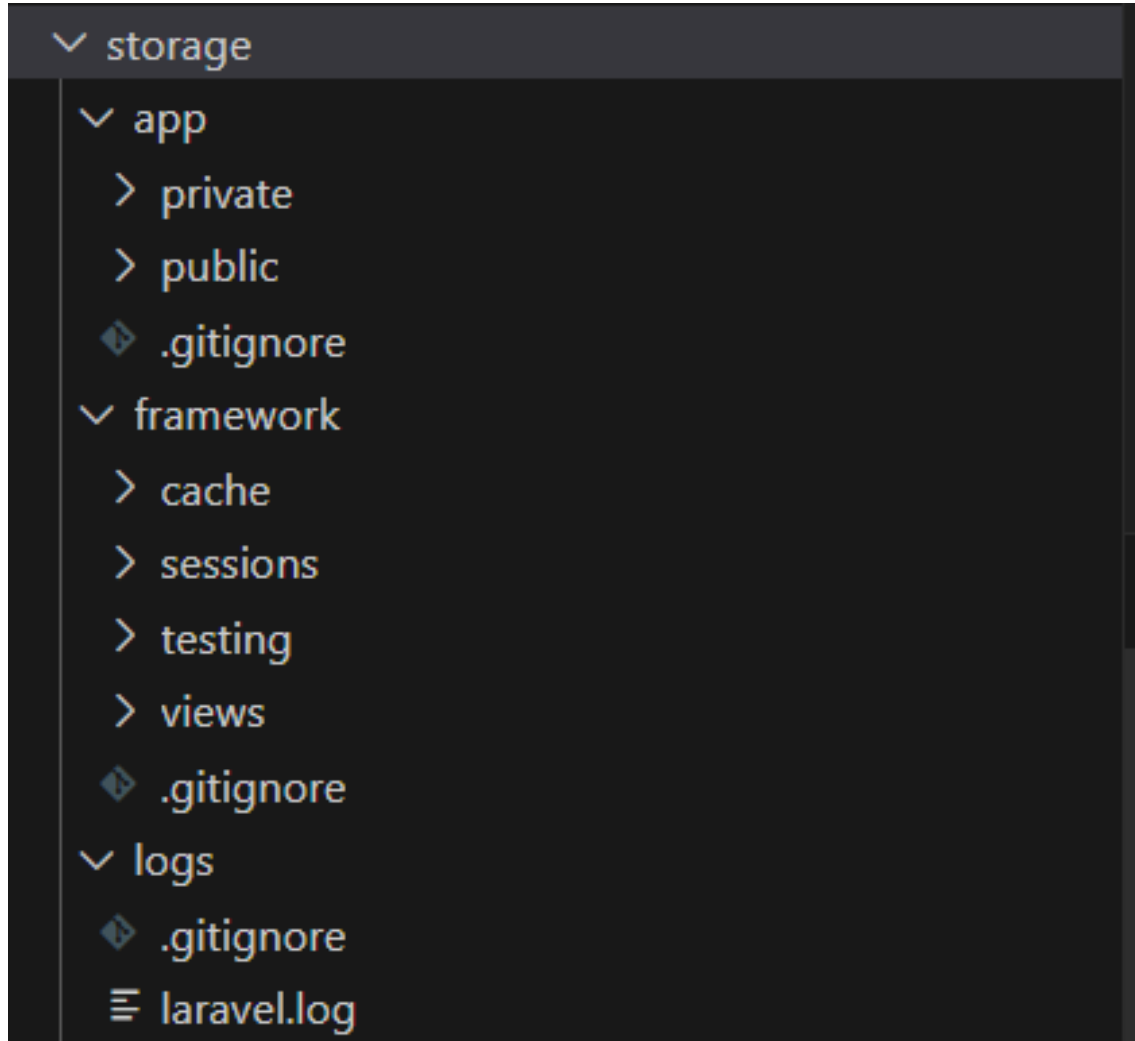
- The **api.php** file contains routes that are intended to be stateless, so requests entering the application through these routes are intended to be authenticated via tokens and will not have access to session state.
- The **channels.php** file is where you may register all of the event broadcasting channels that your application supports.



The Storage Directory

- The storage directory contains your **logs**, **compiled Blade templates**, **file based sessions**, **file caches**, and other files generated by the framework.
- This directory is segregated into **app**, **framework**, and **logs** directories. The app directory may be used to store any files generated by your application.
- The framework directory is used to store framework generated files and caches. Finally, the logs directory contains your application's log files.

Cont. ...



- The `storage/app/public` directory may be used to store user-generated files, such as profile avatars, that should be publicly accessible. You should create a symbolic link at `public/storage` which points to this directory. You may create the link using the php artisan `storage:link` Artisan command.

The Tests Directory

- The tests directory contains your automated tests. Example Pest or PHPUnit unit tests and feature tests are provided out of the box.
- Each test class should be suffixed with the word Test. You may run your tests using the `/vendor/bin/pest` or `/vendor/bin/phpunit` commands.
- Or, if you would like a more detailed and beautiful representation of your test results, you may run your tests using the **`php artisan test`** Artisan command.

Cont. ...

▼ tests

▼ Feature

🐘 ExampleTest.php

▼ Unit

🐘 ExampleTest.php

🐘 Pest.php

🐘 TestCase.php

> vendor

⚙️ .editorconfig

⚙️ .env

💰 .env.example

🔗 .gitattributes

🔗 .gitignore

≡ artisan

{ } composer.json

{ } composer.lock

{ } package-lock.json

{ } package.json

10. vendor/

This directory contains all of the application's third-party dependencies that are installed via Composer.

- **Files:** All third-party libraries, including the Laravel framework itself, are installed here.

Purpose: Manages external libraries and dependencies used by your application.

Node_modules

11. `node_modules/`

This directory contains the front-end JavaScript packages installed via Node.js and NPM (Node Package Manager).

Purpose: Stores frontend development dependencies like Webpack, Vue.js, React, etc.

12. `.env` File

While not a directory, the `.env` file is crucial for setting environment-specific configuration variables.

- This file holds sensitive information like database credentials, API keys, and environment-specific configurations.

Purpose: Stores environment-specific configuration details in a secure and centralized way.

Cont. ...

Summary Table:

| Directory | Description |
|----------------------------|---|
| <code>app/</code> | Contains core application logic like models, controllers, and services. |
| <code>bootstrap/</code> | Manages application bootstrapping and cached configurations. |
| <code>config/</code> | Contains all application configuration files. |
| <code>database/</code> | Manages database migrations, seeders, and factories. |
| <code>public/</code> | Exposes public-facing files like assets and routes requests. |
| <code>resources/</code> | Contains views, language files, and frontend assets. |
| <code>routes/</code> | Defines application routes for web, API, console, and broadcasting. |
| <code>storage/</code> | Stores logs, caches, and other generated files. |
| <code>tests/</code> | Contains unit and feature tests. |
| <code>vendor/</code> | Houses third-party dependencies installed via Composer. |
| <code>node_modules/</code> | Stores JavaScript packages installed via NPM. |
| <code>.env</code> | Stores environment-specific variables and sensitive configurations. |



Thank you!

Appreciate your action.