Laravel Framework

A Powerful PHP Framework for Web Development

This project is funded by the European Union

Maharah
Code Your Future

LibyanSpider

Overview

Laravel is a free, open-source PHP framework that is designed for web application development. It follows the Model-View-Controller (MVC) architectural pattern, providing developers with a robust and elegant toolkit to build efficient and scalable web applications.

Benefits of Laravel

Laravel offers numerous benefits that make it a popular choice among developers.

- **Expressive Syntax**

  Laravel provides an expressive and readable syntax, allowing developers to write clean and concise code.

- **Modularity**

  The framework is highly modular, making it easy to organize and maintain code.

- **Database Abstraction**

  Laravel provides a powerful database abstraction layer, making it effortless to work with various database systems.

- **Security**

  Laravel comes with built-in security features, such as protection against cross-site scripting (XSS) and cross-site request forgery (CSRF) attacks.

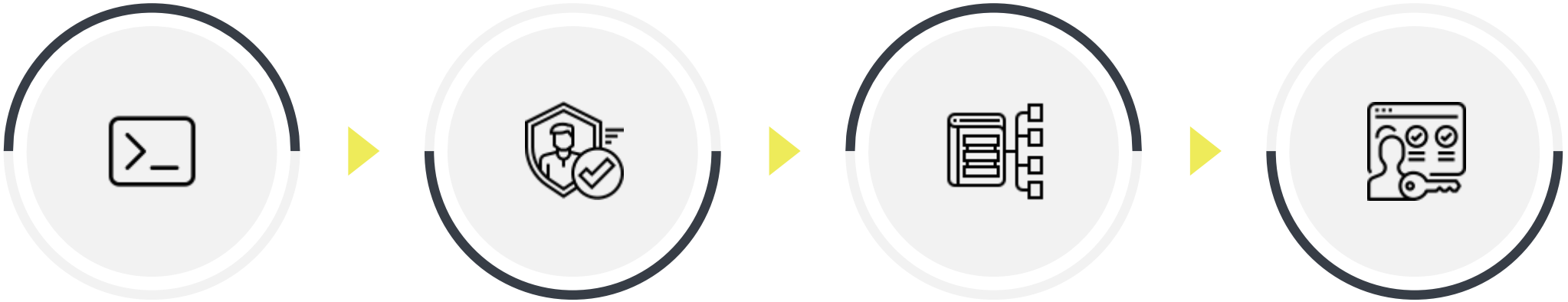
- **Testing and Debugging**

  Laravel offers excellent support for testing and debugging, facilitating the development process.

Features of Laravel

01

Laravel comes equipped with a wide range of features that enhance productivity and streamline development.



Artisan

A command-line interface included with Laravel, providing a number of helpful commands for common tasks such as migrations, testing, and application development.

Authentication

Built-in functionality for handling user authentication, including registration, login, and password reset, making it easy to manage user access.

MVC Architecture

Laravel follows the Model-View-Controller (MVC) pattern, ensuring a clear separation of business logic, UI, and control logic, for clean and maintainable code.

Authorization

Tools for implementing user authorization, such as gates and policies, to control access to resources based on user roles and permissions.

02

Eloquent ORM

An object-relational mapper that allows interaction with databases through expressive, fluent syntax instead of writing SQL queries.

Template Processor

Laravel's templating engine, Blade, provides powerful tools for embedding PHP code into HTML and creating layouts with reusable components.

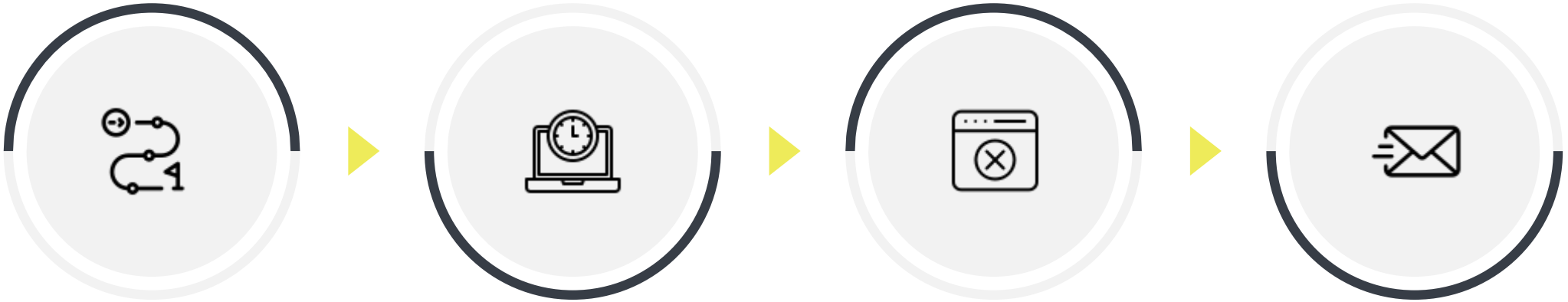
Cache

Built-in caching mechanisms for storing data in a variety of backends (like Memcached and Redis), improving performance by reducing database load.

Queues

System for deferring the processing of time-consuming tasks, such as sending emails, until a later time, improving application responsiveness.

03



Routing

Easy-to-use methods for defining routes in a web application, allowing actions to be linked to different URLs.

Session

Session management for maintaining user state and data between requests, supporting a variety of backend stores.

Error Handling

Integrated error and exception handling, providing a convenient way to log errors and display user-friendly error messages.

Mail

A clean, simple API for sending emails, supporting a variety of drivers and services for easy integration with mail services.

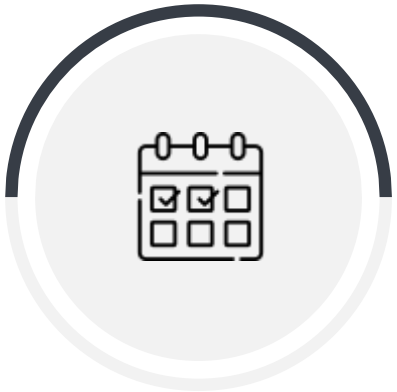
04

Notifications

Simplified API for sending notifications across a variety of delivery channels, including email, SMS, and Slack.

Security

Strong security features, including protection against CSRF, XSS, and SQL injection attacks, ensuring a secure foundation for web applications.



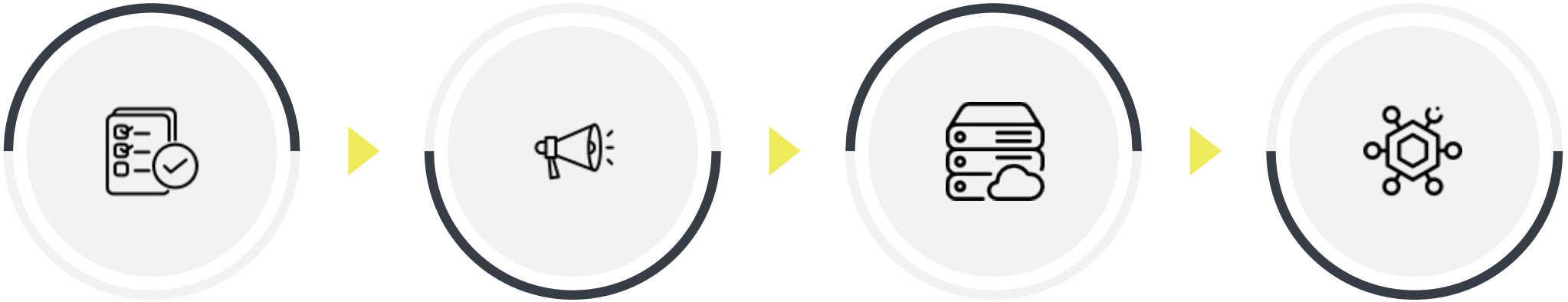
Task Scheduling

Simplifies task scheduling with a concise, expressive syntax, allowing tasks like sending emails or database cleanup to be scheduled effortlessly.

Testing

Built-in support for testing with PHPUnit, offering a convenient way to write and run test cases for your application.

05



Validation

Fluent, convenient validation system for ensuring data integrity by applying various validation rules to request data.

Broadcasting

Real-time event broadcasting for implementing features like live chat or real-time notifications using WebSockets and Laravel Echo.

File Storage

File storage abstraction layer, allowing easy file uploads and management, integrated with cloud storage providers like Amazon S3.

Libraries and Modular

Extensive set of libraries and a modular structure that allows developers to use only the components they need, enhancing application efficiency.

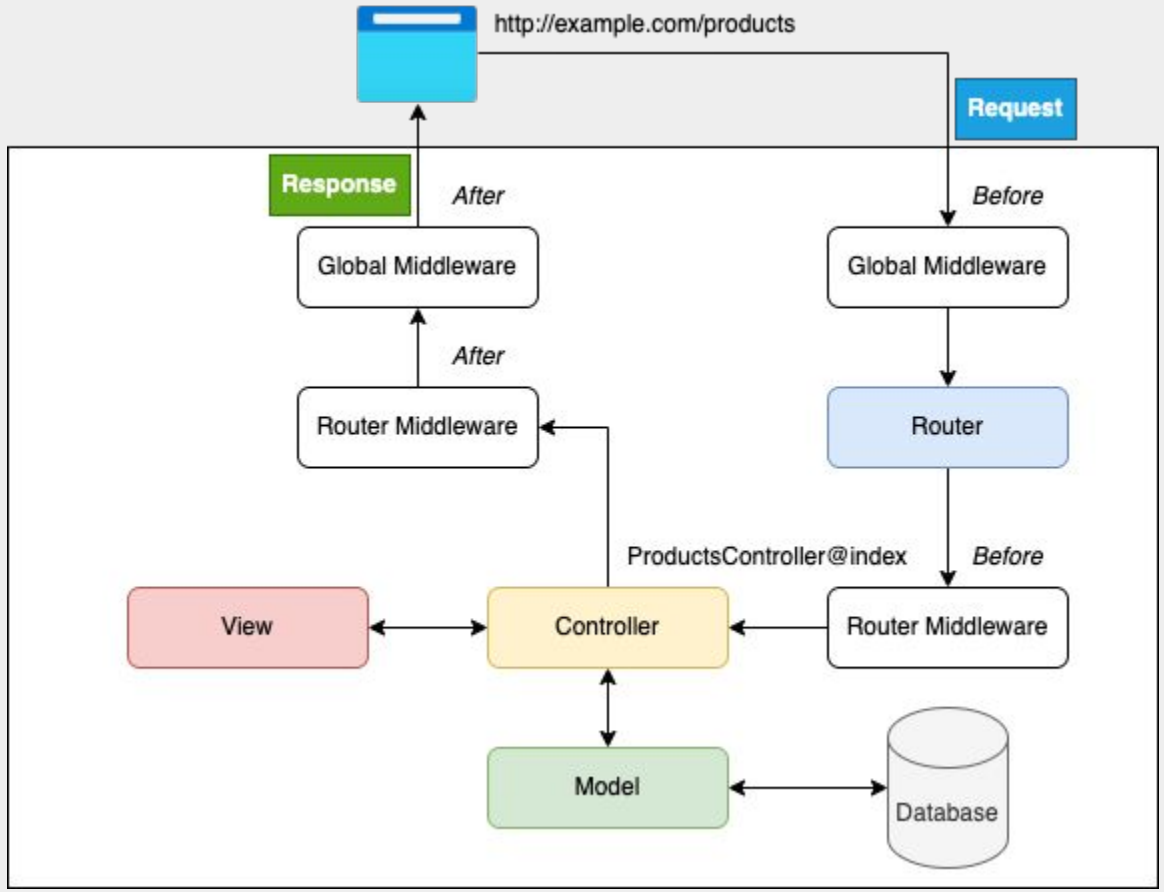
Installation and Setup

1. System Requirements
   - Deployment Documentation > Server Requirements
2. Installing Composer
   - Download and install Composer, a dependency manager for PHP.
   - Official website: getcomposer.org
3. Creating a New Project
   - `composer create-project --prefer-dist laravel/laravel project-name`.
4. Local Development Server
   - Navigate to your project directory in the terminal.
   - Run `php artisan serve` to start the local development server.
   - Access your application at http://localhost:8000.
5. Additional Configuration
   - Environment settings can be modified in the .env file.
   - Configure database connections, mail drivers, and other services as needed.

Request Lifecycle

1. **Entry Point**: Every request to a Laravel application begins at the public/index.php file, acting as the front controller.
2. **Kernel Handling**: The request is sent to either the HTTP or Console kernel, which sets up base services.
3. **Service Providers**: These are bootstrapped, setting up the framework's core services.
4. **Routing**: The request is routed to the appropriate controller and method based on the URL and HTTP verb.
5. **Middleware**: Before reaching the controller, the request may pass through various middleware, which can modify the request or perform checks.
6. **Controller & Response**: The controller handles the request and returns a response, which may be a view, JSON, redirect, etc.
7. **Rendering**: The response is rendered and sent back to the user.

Lifecycle Diagram



Service Providers

Service providers are a crucial component of the Laravel framework. They are responsible for bootstrapping the application, registering bindings, and performing various tasks related to dependency injection and service container management.

- **Role of Service Providers**

  They are central to all Laravel application bootstrapping. Providers configure and prepare services and components.

- **Registration & Booting**

  Service Providers are registered in the `config/app.php` file and bootstrapped by the application.

- **Custom Service Providers**

  You can create your own providers to bootstrap application-specific services or packages.

- **Example**

  Creating a `PaymentServiceProvider` to set up a payment gateway service.

Facades

- **What are Facades?**: In Laravel, facades serve as "static proxies" to underlying classes in the service container, providing a static syntax for accessing methods.
- **Benefits**: They provide a succinct, expressive syntax without losing flexibility and testability.
- **Under the Hood**: Facades use the __callStatic() magic method to defer calls to the resolved service object.
- **Common Facades**: Cache, Route, Auth, etc.

Example:
Using the Cache facade to store data:

```
Cache::put('key', 'value', $minutes);
```

Behind the scenes, it calls the put method on the cache service instance.

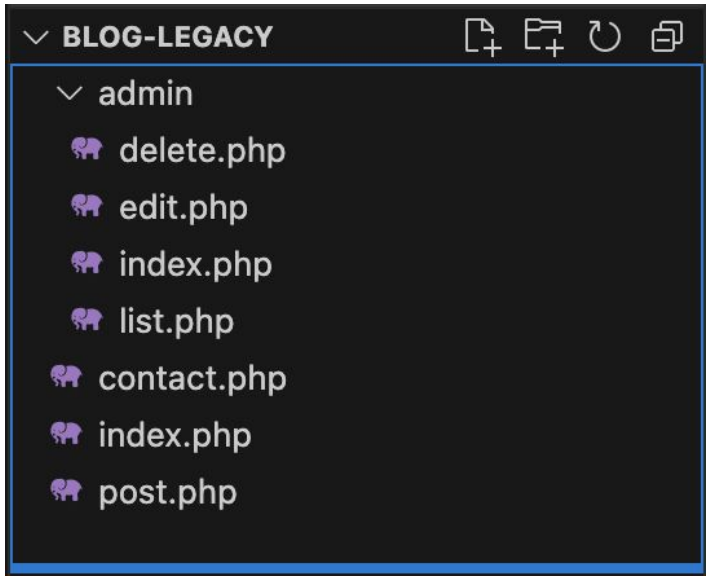
Basic Routing

Routing is a crucial aspect of web development, and Laravel provides a powerful and intuitive routing system. Here are the basics of routing in Laravel:

- **Route Definition**: Routes in Laravel are defined in the routes/web.php file or routes/api.php file. Routes specify the URL pattern and the corresponding action to be executed when that URL is accessed.
- **Route Methods**: Laravel supports various HTTP methods, including GET, POST, PUT, PATCH, DELETE, etc. Developers can define routes using the appropriate method for handling different types of requests.
- **Route Parameters**: Laravel allows you to define route parameters, which capture segments of the URL and pass them as arguments to the corresponding action. Route parameters are denoted by {} in the route definition.

```php
Route::get('/users/{id}', function ($id) {
    // TODO: Logic to fetch user with the specified ID
    return response()->json(['user_id' => $id]);
});
```

Blog Application Routes




```php
<?php

// routes/web.php

use Illuminate\Support\Facades\Route;
use App\Http\Controllers\PostController;
use App\Http\Controllers\AdminPostController;
use App\Http\Controllers\ContactController;

// Public Routes
Route::get('/', [PostController::class, 'index'])->name('posts.index');
Route::get('/posts/{post}', [PostController::class, 'show']);

Route::get('/contact', [ContactController::class, 'create']);
Route::post('/contact', [ContactController::class, 'store']);

// Admin Routes for Post CRUD operations
Route::group(['prefix' => 'admin', 'middleware' => ['auth']], function () {
    Route::get('/posts', [AdminPostController::class, 'index']);
    Route::get('/posts/create', [AdminPostController::class, 'create']);
    Route::post('/posts', [AdminPostController::class, 'store']);
    Route::get('/posts/{post}/edit', [AdminPostController::class, 'edit']);
    Route::put('/posts/{post}', [AdminPostController::class, 'update']);
    Route::delete('/posts/{post}', [AdminPostController::class, 'destroy']);
});
```

Maharah
Code Your Future

Controller Basics

Controllers in Laravel provide a structured and organized way to handle request logic. Here are the basics of working with controllers in Laravel:

- **Controller Creation**: Controllers can be created using the `php artisan make:controller PageController` command. This generates a new controller file in the `app/Http/Controllers` directory.
- **Action Methods**: Controller actions are methods within the controller class that handle specific requests. Each action represents a distinct functionality of the application.
- **Request Handling**: Within the controller actions, you can access and process the incoming request data using the `Illuminate\Http\Request` object.

Controller Example

```php
// routes/web.php
Router::get('users', [UserController::class, 'index']);
Router::get('users/{id}', [UserController::class, 'show']);
Router::post('users', [UserController::class, 'store']);

// app/Http/Controllers/UserController.php
namespace App\Http\Controllers;

use Illuminate\Http\Request;

class UserController extends Controller
{
    public function index()
    {
        // Logic to fetch all users
    }

    public function show($id)
    {
        // Logic to fetch user with the specified ID
    }

    public function store(Request $request)
    {
        // Logic to store a new user based on the request data
    }
}
```

Resource Controllers

Provide a convenient way to define routes and actions for CRUD operations on a resource. Here's how resource controllers work in Laravel:

- **Route Declaration**: Resource routes can be declared using the `Route::resource` method. This method automatically generates the necessary routes for CRUD operations.
- **Controller Binding**: Resource routes are bound to a corresponding controller using the --model option when generating the controller. This associates the routes with the specified model.
- **Standard Actions**: Resource controllers automatically generate standard actions for CRUD operations, including `index`, `create`, `store`, `show`, `edit`, `update`, and `destroy`.

```
Route::resource('users', UserController::class);
```

Laravel Views

Views in Laravel are responsible for rendering the HTML or other response formats that are sent back to the user's browser. Here's how to create views in Laravel:

- **View Files**: Views are typically stored in the `resources/views` directory. Laravel uses the `.blade.php` file extension for its views.
- **View Rendering**: Views can be rendered using the `view()` function, which accepts the name of the view file as its argument. The rendered view can then be returned as a response or assigned to a variable.

```
return view('welcome');
```

Blade Syntax

Blade is Laravel's powerful templating engine, which provides a concise and expressive way to write PHP code in views. Here are some key features of Blade syntax:

- **Echoing Data**: Blade allows you to echo data using the `{{ }}` syntax. This automatically escapes the output to prevent cross-site scripting (XSS) attacks.
- **Conditional Statements**: Blade provides convenient shortcuts for common conditional statements, such as `@if`, `@else`, `@elseif`, and `@unless`.
- **Looping Structures**: Blade simplifies the process of iterating over arrays or collections using `@foreach`, `@for`, and `@while` directives.

```
<p>Welcome, {{ $username }}</p>

@if($isAdmin)
    <p>Welcome, Admin!</p>
@else
    <p>Welcome, User!</p>
@endif

@foreach($users as $user)
    <p>{{ $user->name }}</p>
@endforeach
```

Layouts and Components

Laravel provides powerful features for creating reusable UI components and layouts. These help in organizing and maintaining consistent designs across multiple views.

- **Layouts**

  Layouts act as a wrapper for views and define the common structure and design elements. You can define a layout using the `@extends` directive and include specific sections using `@yield` and `@section`.

- **Components**

  Components allow you to encapsulate reusable UI elements and logic. You can create a component using the `@component` directive and pass data to the component using slots.

Layout Example

```
<!-- layout.blade.php -->
<html>
    <head>
        <title>@yield('title')</title>
    </head>
    <body>
        @yield('content')
    </body>
</html>

<!-- welcome.blade.php -->
@extends('layout')
@section('title', 'Welcome')
@section('content')
    <h1>Welcome to my website!</h1>
@endsection
```

Component Example

```
<!-- alert.blade.php -->
@component('components.alert')
   @slot('type', 'success')
   @slot('message')
       This is a success message.
   @endslot
@endcomponent

<!-- components/alert.blade.php -->
<div class="alert alert-{{ $type }}">
   {{ $message }}
</div>
```

Exercise: Book Rental Library

01

Mission

You are building a book rental library. In this phase, you need to set up the basic pages: a homepage, a page to view books, a page to rent a book, and a page to return a book.