Laravel Packages Scenarios

A comprehensive overview of Laravel package development

# Introduction to Laravel Packages

* **What are Laravel packages?**
* **Why use packages in your projects?**
* **Package ecosystem overview**
* **Types of packages you might create**

# Package Development Basics

* **Structure of a Laravel package**

your-package/  
 ├── src/  
 │ ├── YourPackageServiceProvider.php  
 │ └── Facades/  
 ├── config/  
 │ └── your-package.php  
 ├── database/  
 │ └── migrations/  
 ├── resources/  
 │ ├── views/  
 │ └── lang/  
 ├── routes/  
 │ └── web.php  
 ├── tests/  
 ├── composer.json  
 └── README.md

* **composer.json configuration**

{  
 "name": "your-vendor/your-package",  
 "description": "Your package description",  
 "type": "library",  
 "license": "MIT",  
 "authors": [  
 {  
 "name": "Your Name",  
 "email": "your.email@example.com"  
 }  
 ],  
 "autoload": {  
 "psr-4": {  
 "YourVendor\\YourPackage\\": "src/"  
 }  
 },  
 "require": {  
 "php": "^8.1",  
 "illuminate/support": "^10.0"  
 },  
 "require-dev": {  
 "orchestra/testbench": "^8.0",  
 "phpunit/phpunit": "^10.0"  
 },  
 "extra": {  
 "laravel": {  
 "providers": [  
 "YourVendor\\YourPackage\\YourPackageServiceProvider"  
 ],  
 "aliases": {  
 "YourPackage": "YourVendor\\YourPackage\\Facades\\YourPackage"  
 }  
 }  
 }  
 }

* **Service providers**

namespace YourVendor\YourPackage;  
  
 use Illuminate\Support\ServiceProvider;  
  
 class YourPackageServiceProvider extends ServiceProvider  
 {  
 public function register()  
 {  
 // Register bindings in the container  
 $this->app->singleton('your-package', function ($app) {  
 return new YourPackage($app['config']['your-package']);  
 });  
   
 // Merge configuration  
 $this->mergeConfigFrom(  
 \_\_DIR\_\_.'/../config/your-package.php', 'your-package'  
 );  
 }  
  
 public function boot()  
 {  
 // Publish configuration  
 $this->publishes([  
 \_\_DIR\_\_.'/../config/your-package.php' => config\_path('your-package.php'),  
 ], 'config');  
   
 // Publish migrations  
 $this->publishes([  
 \_\_DIR\_\_.'/../database/migrations/' => database\_path('migrations'),  
 ], 'migrations');  
   
 // Load routes  
 $this->loadRoutesFrom(\_\_DIR\_\_.'/../routes/web.php');  
   
 // Load views  
 $this->loadViewsFrom(\_\_DIR\_\_.'/../resources/views', 'your-package');  
 }  
 }

* **Package auto-discovery**
* **Facades and contracts**

// Contract (Interface)  
 namespace YourVendor\YourPackage\Contracts;  
  
 interface YourPackageInterface  
 {  
 public function doSomething($param);  
 }  
  
 // Implementation  
 namespace YourVendor\YourPackage;  
  
 use YourVendor\YourPackage\Contracts\YourPackageInterface;  
  
 class YourPackage implements YourPackageInterface  
 {  
 public function doSomething($param)  
 {  
 return "Did something with {$param}";  
 }  
 }  
  
 // Facade  
 namespace YourVendor\YourPackage\Facades;  
  
 use Illuminate\Support\Facades\Facade;  
  
 class YourPackage extends Facade  
 {  
 protected static function getFacadeAccessor()  
 {  
 return 'your-package';  
 }  
 }

# Scenario 1: Creating an API Wrapper Package

* **Use case: Simplifying third-party API integration**
* **Structuring API clients**

// API Client  
 namespace YourVendor\PaymentGateway;  
  
 class Client  
 {  
 protected $apiKey;  
 protected $baseUrl;  
 protected $httpClient;  
   
 public function \_\_construct($apiKey, $baseUrl = 'https://api.payment-gateway.com')  
 {  
 $this->apiKey = $apiKey;  
 $this->baseUrl = $baseUrl;  
 $this->httpClient = new \GuzzleHttp\Client([  
 'base\_uri' => $this->baseUrl,  
 'headers' => [  
 'Authorization' => 'Bearer ' . $this->apiKey,  
 'Content-Type' => 'application/json',  
 'Accept' => 'application/json',  
 ]  
 ]);  
 }  
   
 public function createCharge(ChargeRequest $request): ChargeResponse  
 {  
 $response = $this->httpClient->post('/v1/charges', [  
 'json' => $request->toArray(),  
 ]);  
   
 return new ChargeResponse(  
 json\_decode($response->getBody()->getContents(), true)  
 );  
 }  
   
 // Other API methods...  
 }  
   
 // Request DTO  
 class ChargeRequest  
 {  
 public $amount;  
 public $currency;  
 public $description;  
 public $customerId;  
   
 public function toArray()  
 {  
 return [  
 'amount' => $this->amount,  
 'currency' => $this->currency,  
 'description' => $this->description,  
 'customer\_id' => $this->customerId,  
 ];  
 }  
 }

* **Config management**

// config/payment-gateway.php  
 return [  
 'api\_key' => env('PAYMENT\_GATEWAY\_API\_KEY'),  
 'secret' => env('PAYMENT\_GATEWAY\_SECRET'),  
 'environment' => env('PAYMENT\_GATEWAY\_ENV', 'sandbox'),  
 'endpoints' => [  
 'sandbox' => 'https://sandbox.api.payment-gateway.com',  
 'production' => 'https://api.payment-gateway.com',  
 ],  
 'timeout' => 30,  
 'retries' => 3,  
 ];  
   
 // ServiceProvider  
 public function register()  
 {  
 $this->app->singleton('payment-gateway', function ($app) {  
 $config = $app['config']['payment-gateway'];  
   
 // Validate configuration  
 if (empty($config['api\_key'])) {  
 throw new \InvalidArgumentException('Payment gateway API key is required');  
 }  
   
 $baseUrl = $config['endpoints'][$config['environment']];  
   
 return new Client(  
 $config['api\_key'],   
 $baseUrl,   
 $config['timeout'],   
 $config['retries']  
 );  
 });  
 }

* **Rate limiting and caching**

class CachedClient extends Client  
 {  
 protected $cache;  
 protected $rateLimiter;  
   
 public function \_\_construct($apiKey, $baseUrl, Cache $cache, RateLimiter $rateLimiter)  
 {  
 parent::\_\_construct($apiKey, $baseUrl);  
 $this->cache = $cache;  
 $this->rateLimiter = $rateLimiter;  
 }  
   
 public function getCustomer($customerId)  
 {  
 $cacheKey = "customer:{$customerId}";  
   
 return $this->cache->remember($cacheKey, 3600, function () use ($customerId) {  
 // Check rate limit before making request  
 $this->rateLimiter->throttle('api-calls', 60, 100); // 100 calls per minute  
   
 return parent::getCustomer($customerId);  
 });  
 }  
   
 public function createCharge(ChargeRequest $request): ChargeResponse  
 {  
 // Don't cache writes, but still throttle  
 $this->rateLimiter->throttle('api-calls', 60, 100);  
   
 return parent::createCharge($request);  
 }  
 }

* **Error handling**

// Exception hierarchy  
 namespace YourVendor\PaymentGateway\Exceptions;  
   
 class PaymentGatewayException extends \Exception {}  
 class AuthenticationException extends PaymentGatewayException {}  
 class RateLimitException extends PaymentGatewayException {}  
 class ValidationException extends PaymentGatewayException {}  
 class ServerException extends PaymentGatewayException {}  
   
 // In the client  
 protected function handleRequest($method, $endpoint, $options = [])  
 {  
 try {  
 $response = $this->httpClient->$method($endpoint, $options);  
 return json\_decode($response->getBody()->getContents(), true);  
 } catch (\GuzzleHttp\Exception\ClientException $e) {  
 $responseBody = json\_decode($e->getResponse()->getBody()->getContents(), true);  
   
 // Map API error codes to our exceptions  
 switch ($e->getResponse()->getStatusCode()) {  
 case 401:  
 throw new AuthenticationException($responseBody['message'] ?? 'Authentication failed');  
 case 422:  
 throw new ValidationException($responseBody['message'] ?? 'Validation failed', 0, null, $responseBody['errors'] ?? []);  
 case 429:  
 throw new RateLimitException($responseBody['message'] ?? 'Rate limit exceeded');  
 default:  
 throw new PaymentGatewayException($responseBody['message'] ?? 'Unknown error');  
 }  
 } catch (\GuzzleHttp\Exception\ServerException $e) {  
 throw new ServerException('The payment gateway is experiencing issues', 0, $e);  
 }  
 }

# Scenario 2: Authentication & Authorization Packages

* **Custom login providers**

// LDAP Authentication Provider  
 namespace YourVendor\LdapAuth;  
  
 use Illuminate\Contracts\Auth\Authenticatable;  
 use Illuminate\Contracts\Auth\UserProvider;  
  
 class LdapUserProvider implements UserProvider  
 {  
 protected $ldapConnection;  
 protected $model;  
   
 public function \_\_construct($ldapConnection, $model)  
 {  
 $this->ldapConnection = $ldapConnection;  
 $this->model = $model;  
 }  
   
 public function retrieveById($identifier)  
 {  
 return $this->model::find($identifier);  
 }  
   
 public function retrieveByCredentials(array $credentials)  
 {  
 // Find the user by their username in our local database  
 $user = $this->model::where('username', $credentials['username'])->first();  
   
 if (!$user) {  
 // User doesn't exist locally, check if they exist in LDAP  
 $ldapUser = $this->ldapConnection->findUser($credentials['username']);  
   
 if ($ldapUser) {  
 // Create a local user record  
 $user = $this->model::create([  
 'username' => $credentials['username'],  
 'email' => $ldapUser['mail'],  
 'name' => $ldapUser['displayName'],  
 ]);  
 }  
 }  
   
 return $user;  
 }  
   
 public function validateCredentials(Authenticatable $user, array $credentials)  
 {  
 // Authenticate against LDAP  
 return $this->ldapConnection->authenticate(  
 $credentials['username'],   
 $credentials['password']  
 );  
 }  
 }

* **Role-based access control**

// Migration for roles and permissions  
 Schema::create('roles', function (Blueprint $table) {  
 $table->id();  
 $table->string('name');  
 $table->string('label')->nullable();  
 $table->timestamps();  
 });  
  
 Schema::create('permissions', function (Blueprint $table) {  
 $table->id();  
 $table->string('name');  
 $table->string('label')->nullable();  
 $table->timestamps();  
 });  
  
 Schema::create('permission\_role', function (Blueprint $table) {  
 $table->id();  
 $table->foreignId('permission\_id')->constrained()->onDelete('cascade');  
 $table->foreignId('role\_id')->constrained()->onDelete('cascade');  
 $table->timestamps();  
 });  
  
 Schema::create('role\_user', function (Blueprint $table) {  
 $table->id();  
 $table->foreignId('role\_id')->constrained()->onDelete('cascade');  
 $table->foreignId('user\_id')->constrained()->onDelete('cascade');  
 $table->timestamps();  
 });  
   
 // Role trait  
 trait HasRoles  
 {  
 public function roles()  
 {  
 return $this->belongsToMany(Role::class);  
 }  
   
 public function assignRole($role)  
 {  
 if (is\_string($role)) {  
 $role = Role::whereName($role)->firstOrFail();  
 }  
   
 $this->roles()->syncWithoutDetaching($role);  
   
 return $this;  
 }  
   
 public function hasRole($role)  
 {  
 if (is\_string($role)) {  
 return $this->roles->contains('name', $role);  
 }  
   
 return !! $role->intersect($this->roles)->count();  
 }  
 }

* **Permission systems**

// Middleware for permission checks  
 class CheckPermission  
 {  
 public function handle($request, Closure $next, $permission)  
 {  
 // Check if the user has the required permission  
 if (!$request->user() || !$request->user()->can($permission)) {  
 abort(403, 'Unauthorized action.');  
 }  
   
 return $next($request);  
 }  
 }  
   
 // Permission trait  
 trait HasPermissions  
 {  
 public function permissions()  
 {  
 return $this->belongsToMany(Permission::class);  
 }  
   
 public function givePermissionTo($permission)  
 {  
 if (is\_string($permission)) {  
 $permission = Permission::whereName($permission)->firstOrFail();  
 }  
   
 $this->permissions()->syncWithoutDetaching($permission);  
   
 return $this;  
 }  
   
 public function hasDirectPermission($permission)  
 {  
 if (is\_string($permission)) {  
 return $this->permissions->contains('name', $permission);  
 }  
   
 return !! $permission->intersect($this->permissions)->count();  
 }  
   
 public function can($permission)  
 {  
 // Check direct permissions  
 if ($this->hasDirectPermission($permission)) {  
 return true;  
 }  
   
 // Check role permissions  
 foreach ($this->roles as $role) {  
 if ($role->hasPermissionTo($permission)) {  
 return true;  
 }  
 }  
   
 return false;  
 }  
 }

* **Multi-tenancy solutions**

// Global scope for tenant filtering  
 class TenantScope implements Scope  
 {  
 public function apply(Builder $builder, Model $model)  
 {  
 if (auth()->check()) {  
 $builder->where('tenant\_id', auth()->user()->tenant\_id);  
 }  
 }  
 }  
   
 // Model trait  
 trait BelongsToTenant  
 {  
 protected static function booted()  
 {  
 static::addGlobalScope(new TenantScope);  
   
 static::creating(function ($model) {  
 if (auth()->check()) {  
 $model->tenant\_id = auth()->user()->tenant\_id;  
 }  
 });  
 }  
   
 public function tenant()  
 {  
 return $this->belongsTo(Tenant::class);  
 }  
 }

* **SSO implementations**

// JWT Authentication  
 use Firebase\JWT\JWT;  
   
 class JwtAuthGuard implements Guard  
 {  
 protected $request;  
 protected $provider;  
 protected $user;  
   
 public function \_\_construct(Request $request, UserProvider $provider)  
 {  
 $this->request = $request;  
 $this->provider = $provider;  
 }  
   
 public function check()  
 {  
 return ! is\_null($this->user());  
 }  
   
 public function guest()  
 {  
 return ! $this->check();  
 }  
   
 public function user()  
 {  
 if ($this->user) {  
 return $this->user;  
 }  
   
 $token = $this->getTokenFromRequest();  
   
 if (!$token) {  
 return null;  
 }  
   
 try {  
 $decoded = JWT::decode(  
 $token,   
 config('jwt.secret'),   
 ['HS256']  
 );  
   
 $this->user = $this->provider->retrieveById($decoded->sub);  
   
 return $this->user;  
 } catch (\Exception $e) {  
 return null;  
 }  
 }  
   
 protected function getTokenFromRequest()  
 {  
 $header = $this->request->header('Authorization');  
   
 if (Str::startsWith($header, 'Bearer ')) {  
 return Str::substr($header, 7);  
 }  
 }  
 }

# Scenario 3: UI Components & Themes

* **Blade component packages**

// Button component class  
 namespace YourVendor\UiComponents;  
   
 use Illuminate\View\Component;  
   
 class Button extends Component  
 {  
 public $type;  
 public $size;  
 public $color;  
 public $disabled;  
   
 public function \_\_construct(  
 $type = 'button',  
 $size = 'md',  
 $color = 'primary',  
 $disabled = false  
 ) {  
 $this->type = $type;  
 $this->size = $size;  
 $this->color = $color;  
 $this->disabled = $disabled;  
 }  
   
 public function render()  
 {  
 return view('ui-components::button');  
 }  
   
 public function classes()  
 {  
 return [  
 'btn',  
 'btn-' . $this->color,  
 'btn-' . $this->size,  
 $this->disabled ? 'disabled' : '',  
 ];  
 }  
 }  
   
 // Button component view (button.blade.php)  
 <button  
 type="{{ $type }}"  
 {{ $attributes->merge(['class' => implode(' ', $classes())]) }}  
 {{ $disabled ? 'disabled' : '' }}  
 >  
 {{ $slot }}  
 </button>  
   
 // Usage in application  
 <x-ui::button  
 type="submit"  
 color="success"  
 size="lg"  
 class="mt-4 w-full"  
 >  
 Save Changes  
 </x-ui::button>

* **Admin panel packages**

// Admin panel resource definition  
 namespace App\Admin\Resources;  
   
 use YourVendor\AdminPanel\Resources\Resource;  
   
 class UserResource extends Resource  
 {  
 public static $model = 'App\Models\User';  
   
 public function fields()  
 {  
 return [  
 ID::make()->sortable(),  
   
 Text::make('Name')  
 ->sortable()  
 ->rules('required', 'max:255'),  
   
 Email::make('Email')  
 ->sortable()  
 ->rules('required', 'email', 'max:255')  
 ->creationRules('unique:users,email')  
 ->updateRules('unique:users,email,{{resourceId}}'),  
   
 Password::make('Password')  
 ->onlyOnForms()  
 ->creationRules('required', 'string', 'min:8')  
 ->updateRules('nullable', 'string', 'min:8'),  
   
 BelongsToMany::make('Roles'),  
 ];  
 }  
   
 public function cards()  
 {  
 return [  
 new UserStats,  
 new NewUsers,  
 ];  
 }  
   
 public function filters()  
 {  
 return [  
 new RoleFilter,  
 ];  
 }  
   
 public function actions()  
 {  
 return [  
 new ResetPassword,  
 new DeactivateUsers,  
 ];  
 }  
 }

* **Theme systems**

// Theme manager service  
 namespace YourVendor\ThemeManager;  
   
 class ThemeManager  
 {  
 protected $theme;  
 protected $themes = [];  
 protected $defaultTheme;  
   
 public function \_\_construct(array $themes, $defaultTheme)  
 {  
 $this->themes = $themes;  
 $this->defaultTheme = $defaultTheme;  
 $this->theme = $defaultTheme;  
 }  
   
 public function setTheme($theme)  
 {  
 if (!isset($this->themes[$theme])) {  
 throw new \InvalidArgumentException("Theme [{$theme}] not found.");  
 }  
   
 $this->theme = $theme;  
   
 return $this;  
 }  
   
 public function current()  
 {  
 return $this->theme;  
 }  
   
 public function path($file = '')  
 {  
 $themePath = $this->themes[$this->theme]['path'];  
   
 return $file ? $themePath . '/' . $file : $themePath;  
 }  
   
 public function asset($file)  
 {  
 return asset($this->path('assets/' . $file));  
 }  
   
 public function viewNamespace()  
 {  
 return $this->themes[$this->theme]['namespace'];  
 }  
 }  
   
 // Theme service provider  
 class ThemeServiceProvider extends ServiceProvider  
 {  
 public function register()  
 {  
 $this->app->singleton('theme', function ($app) {  
 return new ThemeManager(  
 config('themes.themes'),  
 config('themes.default')  
 );  
 });  
 }  
   
 public function boot()  
 {  
 // Register theme view namespaces  
 $themes = config('themes.themes');  
   
 foreach ($themes as $name => $theme) {  
 $this->loadViewsFrom($theme['path'] . '/views', $theme['namespace']);  
 }  
   
 // Register blade directives  
 Blade::directive('theme', function ($expression) {  
 return "<?php echo theme()->asset($expression); ?>";  
 });  
 }  
 }

* **Asset management**

// vite.config.js  
 import { defineConfig } from 'vite';  
 import laravel from 'laravel-vite-plugin';  
   
 export default defineConfig({  
 plugins: [  
 laravel({  
 input: [  
 'resources/css/app.css',  
 'resources/js/app.js',  
 // Theme assets  
 'resources/themes/default/css/theme.scss',  
 'resources/themes/default/js/theme.js',  
 'resources/themes/dark/css/theme.scss',  
 'resources/themes/dark/js/theme.js',  
 ],  
 refresh: true,  
 }),  
 ],  
 resolve: {  
 alias: {  
 '@': '/resources/js',  
 '@css': '/resources/css',  
 '@themes': '/resources/themes',  
 },  
 },  
 });  
   
 // PHP asset helper (in theme manager)  
 public function assetUrl($path, $secure = null)  
 {  
 $manifestPath = public\_path('build/manifest.json');  
   
 static $manifest;  
   
 if (!$manifest && file\_exists($manifestPath)) {  
 $manifest = json\_decode(file\_get\_contents($manifestPath), true);  
 }  
   
 $themePath = "resources/themes/{$this->theme}/";  
 $assetPath = $themePath . $path;  
   
 if (isset($manifest[$assetPath])) {  
 return asset('build/' . $manifest[$assetPath], $secure);  
 }  
   
 return asset($path, $secure);  
 }

* **Livewire/Inertia component libraries**

// Livewire data table component  
 namespace YourVendor\LiveComponents;  
   
 use Livewire\Component;  
   
 class DataTable extends Component  
 {  
 public $model;  
 public $columns = [];  
 public $search = '';  
 public $perPage = 15;  
 public $sortField = 'id';  
 public $sortDirection = 'asc';  
   
 protected $listeners = ['refresh' => '$refresh'];  
   
 public function sortBy($field)  
 {  
 if ($this->sortField === $field) {  
 $this->sortDirection = $this->sortDirection === 'asc' ? 'desc' : 'asc';  
 } else {  
 $this->sortField = $field;  
 $this->sortDirection = 'asc';  
 }  
 }  
   
 public function updatingSearch()  
 {  
 $this->resetPage();  
 }  
   
 public function getRowsQueryProperty()  
 {  
 $query = $this->model::query();  
   
 if ($this->search) {  
 $query->where(function($subQuery) {  
 foreach ($this->columns as $column) {  
 if (isset($column['searchable']) && $column['searchable']) {  
 $subQuery->orWhere($column['field'], 'like', '%' . $this->search . '%');  
 }  
 }  
 });  
 }  
   
 return $query->orderBy($this->sortField, $this->sortDirection);  
 }  
   
 public function getRowsProperty()  
 {  
 return $this->rowsQuery->paginate($this->perPage);  
 }  
   
 public function render()  
 {  
 return view('live-components::data-table', [  
 'items' => $this->rows,  
 ]);  
 }  
 }

# Scenario 4: Database & Eloquent Extensions

* **Custom database connections**

// Example of read/write connection setup in config/database.php  
 'mysql' => [  
 'read' => [  
 'host' => [  
 env('DB\_READ\_HOST\_1', '192.168.1.1'),  
 env('DB\_READ\_HOST\_2', '192.168.1.2'),  
 ],  
 ],  
 'write' => [  
 'host' => env('DB\_WRITE\_HOST', '192.168.1.3'),  
 ],  
 'sticky' => true,  
 'driver' => 'mysql',  
 'database' => env('DB\_DATABASE', 'forge'),  
 'username' => env('DB\_USERNAME', 'forge'),  
 'password' => env('DB\_PASSWORD', ''),  
 'charset' => 'utf8mb4',  
 'collation' => 'utf8mb4\_unicode\_ci',  
 'prefix' => '',  
 ],  
   
 // Implementing a custom database connection  
 namespace YourVendor\DatabaseExtensions;  
   
 use Illuminate\Database\Connection;  
   
 class CustomConnection extends Connection  
 {  
 public function \_\_construct($pdo, $database = '', $tablePrefix = '', array $config = [])  
 {  
 parent::\_\_construct($pdo, $database, $tablePrefix, $config);  
   
 // Custom initialization for your connection  
 }  
   
 protected function getDefaultQueryGrammar()  
 {  
 return new CustomQueryGrammar();  
 }  
   
 protected function getDefaultSchemaGrammar()  
 {  
 return new CustomSchemaGrammar();  
 }  
 }  
   
 // Register in service provider  
 $this->app->singleton('db.connector.custom', function () {  
 return new CustomConnector();  
 });  
   
 $this->app->singleton('db.connection.custom', function ($app, $config) {  
 return new CustomConnection(  
 $config['pdo'],  
 $config['database'],  
 $config['prefix'],  
 $config  
 );  
 });

* **Eloquent model traits**

// UUID trait example  
 namespace YourVendor\EloquentExtensions\Traits;  
   
 use Illuminate\Support\Str;  
   
 trait HasUuid  
 {  
 protected static function bootHasUuid()  
 {  
 static::creating(function ($model) {  
 if (! $model->{$model->getUuidColumn()}) {  
 $model->{$model->getUuidColumn()} = (string) Str::uuid();  
 }  
 });  
 }  
   
 public function getUuidColumn()  
 {  
 return $this->uuidColumn ?? 'uuid';  
 }  
   
 public function scopeWhereUuid($query, $uuid)  
 {  
 return $query->where($this->getUuidColumn(), $uuid);  
 }  
 }  
   
 // Auditable trait for tracking changes  
 trait Auditable  
 {  
 public static function bootAuditable()  
 {  
 static::created(function ($model) {  
 $model->recordActivity('created');  
 });  
   
 static::updated(function ($model) {  
 $model->recordActivity('updated');  
 });  
   
 static::deleted(function ($model) {  
 $model->recordActivity('deleted');  
 });  
 }  
   
 protected function recordActivity($event)  
 {  
 Activity::create([  
 'user\_id' => auth()->id() ?? null,  
 'model\_type' => get\_class($this),  
 'model\_id' => $this->getKey(),  
 'event' => $event,  
 'changes' => $this->getChanges(),  
 'ip\_address' => request()->ip(),  
 'user\_agent' => request()->userAgent(),  
 ]);  
 }  
 }

* **Query builder extensions**

// Register query builder macros in service provider  
 use Illuminate\Database\Query\Builder;  
 use Illuminate\Support\ServiceProvider;  
   
 class QueryExtensionServiceProvider extends ServiceProvider  
 {  
 public function boot()  
 {  
 // Add a "whereLike" macro for easier "like" searches  
 Builder::macro('whereLike', function ($attributes, $searchTerm) {  
 $this->where(function ($query) use ($attributes, $searchTerm) {  
 foreach (array\_wrap($attributes) as $attribute) {  
 $query->orWhere($attribute, 'LIKE', "%{$searchTerm}%");  
 }  
 });  
   
 return $this;  
 });  
   
 // Add a "whereJsonContains" macro for JSON queries  
 Builder::macro('whereJsonContains', function ($column, $value) {  
 return $this->whereRaw(  
 "JSON\_CONTAINS({$this->grammar->wrap($column)}, ?)",  
 [json\_encode($value)]  
 );  
 });  
   
 // Add a "toRawSql" macro to help with debugging  
 Builder::macro('toRawSql', function () {  
 $bindings = $this->getBindings();  
 $sql = $this->toSql();  
   
 foreach ($bindings as $binding) {  
 $value = is\_numeric($binding) ? $binding : "'".$binding."'";  
 $sql = preg\_replace('/\?/', $value, $sql, 1);  
 }  
   
 return $sql;  
 });  
 }  
 }

* **Migration packages**

// Example of a migration class with safe methods for zero-downtime deployment  
 namespace YourVendor\MigrationPatterns;  
   
 use Illuminate\Database\Schema\Blueprint;  
 use Illuminate\Support\Facades\Schema;  
   
 class SafeMigration  
 {  
 /\*\*  
 \* Add a column safely in multiple steps to avoid locking the table.  
 \*/  
 public static function addColumn($table, $column, $type, $options = [])  
 {  
 // Step 1: Add the column as nullable regardless of final state  
 Schema::table($table, function (Blueprint $table) use ($column, $type, $options) {  
 $table->$type($column)->nullable()->default(null);  
 });  
   
 // Step 2: If the column should not be nullable or has a default value, update it  
 if (isset($options['nullable']) && $options['nullable'] === false) {  
 // First populate any existing rows  
 DB::table($table)->whereNull($column)->update([  
 $column => $options['default'] ?? null,  
 ]);  
   
 // Then update the column to be non-nullable  
 Schema::table($table, function (Blueprint $table) use ($column) {  
 $table->$type($column)->nullable(false)->change();  
 });  
 }  
   
 // Step 3: Set the final default if needed (separate operation)  
 if (isset($options['default'])) {  
 Schema::table($table, function (Blueprint $table) use ($column, $type, $options) {  
 $table->$type($column)  
 ->default($options['default'])  
 ->change();  
 });  
 }  
 }  
 }

* **Database tools & utilities**

// Data anonymizer example  
 namespace YourVendor\DatabaseTools;  
   
 class DataAnonymizer  
 {  
 protected $faker;  
 protected $rules = [];  
   
 public function \_\_construct(\Faker\Generator $faker)  
 {  
 $this->faker = $faker;  
 }  
   
 public function setRules(array $rules)  
 {  
 $this->rules = $rules;  
 return $this;  
 }  
   
 public function anonymize($model, $applyRules = true)  
 {  
 $table = $model->getTable();  
 $primaryKey = $model->getKeyName();  
   
 if (!isset($this->rules[$table]) && $applyRules) {  
 throw new \InvalidArgumentException("No anonymization rules for table {$table}");  
 }  
   
 $rules = $applyRules ? $this->rules[$table] : [];  
   
 $query = DB::table($table);  
   
 // Process in chunks to avoid memory issues  
 $query->orderBy($primaryKey)->chunk(1000, function ($records) use ($table, $rules) {  
 foreach ($records as $record) {  
 $updates = [];  
   
 foreach ($rules as $column => $rule) {  
 $updates[$column] = $this->processRule($rule);  
 }  
   
 if (!empty($updates)) {  
 DB::table($table)  
 ->where('id', $record->id)  
 ->update($updates);  
 }  
 }  
 });  
 }  
   
 protected function processRule($rule)  
 {  
 if (is\_callable($rule)) {  
 return $rule($this->faker);  
 }  
   
 if (is\_string($rule) && method\_exists($this->faker, $rule)) {  
 return $this->faker->$rule();  
 }  
   
 return $rule;  
 }  
 }  
   
 // Using the anonymizer  
 $anonymizer = new DataAnonymizer(Faker\Factory::create());  
   
 $anonymizer->setRules([  
 'users' => [  
 'email' => function ($faker) {  
 return $faker->safeEmail();  
 },  
 'name' => 'name',  
 'phone' => 'phoneNumber',  
 'address' => 'address',  
 'password' => bcrypt('password'),  
 // Keep some fields like id, created\_at, etc.  
 ],  
 // Rules for other tables  
 ]);  
   
 $anonymizer->anonymize(new \App\Models\User());

# Scenario 5: Notification & Communication Packages

* **Custom notification channels**

// Custom notification channel for SMS  
 namespace YourVendor\SmsNotifications;  
   
 use Illuminate\Notifications\Notification;  
   
 class SmsChannel  
 {  
 protected $client;  
   
 public function \_\_construct(SmsClient $client)  
 {  
 $this->client = $client;  
 }  
   
 public function send($notifiable, Notification $notification)  
 {  
 if (!method\_exists($notification, 'toSms')) {  
 throw new \Exception('Notification class must have a toSms method');  
 }  
   
 // Get the phone number from the notifiable entity  
 $to = $notifiable->routeNotificationFor('sms', $notification);  
   
 if (empty($to)) {  
 return;  
 }  
   
 // Get the notification content  
 $message = $notification->toSms($notifiable);  
   
 // Send the SMS  
 $this->client->send($to, $message);  
 }  
 }  
   
 // Using the custom channel in a notification  
 class OrderShipped extends Notification  
 {  
 public function via($notifiable)  
 {  
 return ['mail', 'sms', 'database'];  
 }  
   
 public function toSms($notifiable)  
 {  
 return "Your order #{$this->order->id} has been shipped and will arrive on {$this->order->estimated\_delivery}.";  
 }  
 }

* **Chat & messaging systems**

// Chat event class  
 namespace YourVendor\ChatSystem\Events;  
   
 use Illuminate\Broadcasting\InteractsWithSockets;  
 use Illuminate\Broadcasting\PresenceChannel;  
 use Illuminate\Contracts\Broadcasting\ShouldBroadcast;  
 use Illuminate\Foundation\Events\Dispatchable;  
 use Illuminate\Queue\SerializesModels;  
   
 class MessageSent implements ShouldBroadcast  
 {  
 use Dispatchable, InteractsWithSockets, SerializesModels;  
   
 public $message;  
 public $user;  
 public $conversation;  
   
 public function \_\_construct($user, $message, $conversation)  
 {  
 $this->user = $user;  
 $this->message = $message;  
 $this->conversation = $conversation;  
 }  
   
 public function broadcastOn()  
 {  
 return new PresenceChannel('conversation.' . $this->conversation->id);  
 }  
   
 public function broadcastWith()  
 {  
 return [  
 'id' => $this->message->id,  
 'content' => $this->message->content,  
 'created\_at' => $this->message->created\_at->toIso8601String(),  
 'user' => [  
 'id' => $this->user->id,  
 'name' => $this->user->name,  
 'avatar' => $this->user->profile\_photo\_url,  
 ],  
 ];  
 }  
 }  
   
 // WebSocket channel authorization  
 Broadcast::channel('conversation.{conversationId}', function ($user, $conversationId) {  
 $conversation = Conversation::findOrFail($conversationId);  
 return $conversation->participants->contains('id', $user->id);  
 });

* **Email template management**

// Email template manager  
 namespace YourVendor\EmailTemplates;  
   
 class TemplateManager  
 {  
 protected $templatePath;  
 protected $defaultTheme;  
 protected $cache;  
   
 public function \_\_construct($templatePath, $defaultTheme, $cache)  
 {  
 $this->templatePath = $templatePath;  
 $this->defaultTheme = $defaultTheme;  
 $this->cache = $cache;  
 }  
   
 public function render($template, $data = [], $theme = null, $locale = null)  
 {  
 // Determine theme and locale  
 $theme = $theme ?? $this->defaultTheme;  
 $locale = $locale ?? app()->getLocale();  
   
 // Build the template path  
 $path = "{$this->templatePath}/{$theme}/{$locale}/{$template}.blade.php";  
   
 // Check if the template exists, fall back to default theme or locale if needed  
 if (!file\_exists($path)) {  
 // Try the default theme with the requested locale  
 $path = "{$this->templatePath}/{$this->defaultTheme}/{$locale}/{$template}.blade.php";  
   
 if (!file\_exists($path)) {  
 // Finally, fall back to default theme and locale  
 $path = "{$this->templatePath}/{$this->defaultTheme}/en/{$template}.blade.php";  
 }  
 }  
   
 // Render the template with given data using Laravel's Blade compiler  
 $cacheKey = "email\_template:{$theme}:{$locale}:{$template}";  
   
 return $this->cache->remember($cacheKey, 3600, function () use ($path, $data) {  
 $blade = app('blade.compiler');  
 $rendered = $blade->compileString(file\_get\_contents($path));  
   
 // Extract rendered content and evaluate with data  
 return view()->file($rendered, $data)->render();  
 });  
 }  
 }  
   
 // Using the template manager in a Mailablle  
 class OrderConfirmation extends Mailable  
 {  
 protected $order;  
 protected $user;  
   
 public function \_\_construct(Order $order, User $user)  
 {  
 $this->order = $order;  
 $this->user = $user;  
 }  
   
 public function build()  
 {  
 $template = app(TemplateManager::class);  
 $html = $template->render('order-confirmation', [  
 'order' => $this->order,  
 'user' => $this->user,  
 ], $this->user->email\_theme, $this->user->locale);  
   
 return $this->subject('Order Confirmation #' . $this->order->id)  
 ->html($html);  
 }  
 }

* **SMS/WhatsApp integration**

// SMS message gateway with failover  
 namespace YourVendor\SmsIntegration;  
   
 class SmsGateway  
 {  
 protected $providers = [];  
 protected $defaultProvider;  
   
 public function \_\_construct(array $providers, $defaultProvider)  
 {  
 $this->providers = $providers;  
 $this->defaultProvider = $defaultProvider;  
 }  
   
 public function send($to, $message, $options = [])  
 {  
 $provider = $options['provider'] ?? $this->defaultProvider;  
 $attempt = 0;  
 $maxAttempts = count($this->providers);  
   
 // Try sending with selected provider, then failover if needed  
 while ($attempt < $maxAttempts) {  
 try {  
 $result = $this->sendWithProvider($provider, $to, $message, $options);  
   
 // Track the delivery  
 $this->trackDelivery($provider, $to, $message, $result);  
   
 return $result;  
 } catch (\Exception $e) {  
 // Log the failure  
 logger()->error("SMS delivery failed with provider {$provider}", [  
 'error' => $e->getMessage(),  
 'to' => $to,  
 ]);  
   
 // Move to the next provider  
 $attempt++;  
 $provider = array\_keys($this->providers)[$attempt % count($this->providers)];  
 }  
 }  
   
 throw new \Exception("Failed to send SMS after {$maxAttempts} attempts");  
 }  
   
 protected function sendWithProvider($provider, $to, $message, $options)  
 {  
 if (!isset($this->providers[$provider])) {  
 throw new \InvalidArgumentException("Provider {$provider} not configured");  
 }  
   
 $client = $this->providers[$provider];  
   
 return $client->sendMessage([  
 'to' => $to,  
 'text' => $message,  
 'options' => $options,  
 ]);  
 }  
   
 protected function trackDelivery($provider, $to, $message, $result)  
 {  
 return SmsLog::create([  
 'provider' => $provider,  
 'to' => $to,  
 'message' => $message,  
 'message\_id' => $result['message\_id'] ?? null,  
 'status' => $result['status'] ?? 'sent',  
 'cost' => $result['cost'] ?? null,  
 'meta' => $result,  
 ]);  
 }  
 }

* **Webhook systems**

// Webhook dispatcher  
 namespace YourVendor\WebhookSystem;  
   
 class WebhookDispatcher  
 {  
 protected $queue;  
 protected $secret;  
   
 public function \_\_construct($queue, $secret)  
 {  
 $this->queue = $queue;  
 $this->secret = $secret;  
 }  
   
 public function dispatch($event, $payload, $webhooks)  
 {  
 foreach ($webhooks as $webhook) {  
 $this->sendWebhook($webhook, $event, $payload);  
 }  
 }  
   
 protected function sendWebhook($webhook, $event, $payload)  
 {  
 // Add webhook job to queue  
 $this->queue->push(new DeliverWebhookJob(  
 $webhook->url,  
 $event,  
 $payload,  
 $this->buildHeaders($webhook, $payload),  
 $webhook->id  
 ));  
 }  
   
 protected function buildHeaders($webhook, $payload)  
 {  
 $signature = hash\_hmac('sha256', json\_encode($payload), $this->secret);  
   
 return [  
 'Content-Type' => 'application/json',  
 'X-Webhook-Signature' => $signature,  
 'X-Webhook-Event' => $event,  
 'X-Webhook-Delivery' => Str::uuid()->toString(),  
 ];  
 }  
 }  
   
 // Webhook delivery job  
 class DeliverWebhookJob implements ShouldQueue  
 {  
 use Dispatchable, InteractsWithQueue, Queueable, SerializesModels;  
   
 public $tries = 3;  
 public $backoff = [30, 60, 120]; // Retry after 30s, 60s, 120s  
   
 protected $url;  
 protected $event;  
 protected $payload;  
 protected $headers;  
 protected $webhookId;  
   
 public function \_\_construct($url, $event, $payload, $headers, $webhookId)  
 {  
 $this->url = $url;  
 $this->event = $event;  
 $this->payload = $payload;  
 $this->headers = $headers;  
 $this->webhookId = $webhookId;  
 }  
   
 public function handle()  
 {  
 try {  
 $client = new \GuzzleHttp\Client();  
 $response = $client->post($this->url, [  
 'json' => $this->payload,  
 'headers' => $this->headers,  
 'timeout' => 15.0,  
 ]);  
   
 // Log successful delivery  
 WebhookDelivery::create([  
 'webhook\_id' => $this->webhookId,  
 'event' => $this->event,  
 'payload' => $this->payload,  
 'status' => $response->getStatusCode(),  
 'response' => $response->getBody()->getContents(),  
 'delivered\_at' => now(),  
 ]);  
 } catch (\Exception $e) {  
 // Log failed delivery  
 WebhookDelivery::create([  
 'webhook\_id' => $this->webhookId,  
 'event' => $this->event,  
 'payload' => $this->payload,  
 'status' => $e->getCode(),  
 'response' => $e->getMessage(),  
 'delivered\_at' => null,  
 ]);  
   
 // Rethrow to trigger retry  
 throw $e;  
 }  
 }  
 }

# Package Testing Strategies

* **PHPUnit testing setup**

// phpunit.xml configuration for a package  
 <?xml version="1.0" encoding="UTF-8"?>  
 <phpunit  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 bootstrap="vendor/autoload.php"  
 backupGlobals="false"  
 colors="true"  
 processIsolation="false"  
 stopOnFailure="false"  
 xsi:noNamespaceSchemaLocation="https://schema.phpunit.de/10.0/phpunit.xsd"  
 cacheDirectory=".phpunit.cache"  
 >  
 <coverage>  
 <include>  
 <directory suffix=".php">src/</directory>  
 </include>  
 <exclude>  
 <directory suffix=".php">src/config</directory>  
 </exclude>  
 <report>  
 <clover outputFile="build/logs/clover.xml"/>  
 <html outputDirectory="build/coverage"/>  
 </report>  
 </coverage>  
 <testsuites>  
 <testsuite name="Unit">  
 <directory suffix="Test.php">./tests/Unit</directory>  
 </testsuite>  
 <testsuite name="Feature">  
 <directory suffix="Test.php">./tests/Feature</directory>  
 </testsuite>  
 </testsuites>  
 <php>  
 <env name="APP\_ENV" value="testing"/>  
 <env name="DB\_CONNECTION" value="testing"/>  
 <env name="CACHE\_DRIVER" value="array"/>  
 </php>  
 </phpunit>

* **Orchestra Testbench**

// Base test case using Orchestra Testbench  
 namespace YourVendor\YourPackage\Tests;  
   
 use Orchestra\Testbench\TestCase as BaseTestCase;  
 use YourVendor\YourPackage\YourPackageServiceProvider;  
   
 abstract class TestCase extends BaseTestCase  
 {  
 protected function getPackageProviders($app)  
 {  
 return [  
 YourPackageServiceProvider::class,  
 ];  
 }  
   
 protected function getPackageAliases($app)  
 {  
 return [  
 'YourFacade' => 'YourVendor\YourPackage\Facades\YourFacade',  
 ];  
 }  
   
 protected function getEnvironmentSetUp($app)  
 {  
 // Setup default database  
 $app['config']->set('database.default', 'testbench');  
 $app['config']->set('database.connections.testbench', [  
 'driver' => 'sqlite',  
 'database' => ':memory:',  
 'prefix' => '',  
 ]);  
   
 // Package configuration  
 $app['config']->set('your-package.key', 'value');  
 }  
   
 protected function setUpTraits()  
 {  
 parent::setUpTraits();  
   
 $uses = array\_flip(class\_uses\_recursive(static::class));  
   
 if (isset($uses[RefreshDatabase::class])) {  
 $this->loadMigrationsFrom(\_\_DIR\_\_ . '/../database/migrations');  
 }  
 }  
 }

* **Feature vs. Unit tests**

// Unit test example  
 namespace YourVendor\YourPackage\Tests\Unit;  
   
 use PHPUnit\Framework\TestCase;  
 use YourVendor\YourPackage\Calculator;  
   
 class CalculatorTest extends TestCase  
 {  
 public function test\_it\_adds\_two\_numbers()  
 {  
 $calculator = new Calculator();  
 $result = $calculator->add(5, 10);  
   
 $this->assertEquals(15, $result);  
 }  
   
 public function test\_it\_subtracts\_two\_numbers()  
 {  
 $calculator = new Calculator();  
 $result = $calculator->subtract(15, 10);  
   
 $this->assertEquals(5, $result);  
 }  
 }  
   
 // Feature test example  
 namespace YourVendor\YourPackage\Tests\Feature;  
   
 use YourVendor\YourPackage\Tests\TestCase;  
 use YourVendor\YourPackage\Models\Post;  
 use Illuminate\Foundation\Testing\RefreshDatabase;  
   
 class PostControllerTest extends TestCase  
 {  
 use RefreshDatabase;  
   
 public function test\_it\_creates\_a\_post()  
 {  
 $response = $this->postJson('/api/posts', [  
 'title' => 'Test Title',  
 'body' => 'Test content here',  
 ]);  
   
 $response->assertStatus(201)  
 ->assertJsonStructure([  
 'id',  
 'title',  
 'body',  
 'created\_at',  
 ]);  
   
 $this->assertDatabaseHas('posts', [  
 'title' => 'Test Title',  
 'body' => 'Test content here',  
 ]);  
 }  
 }

* **Testing with different Laravel versions**

// composer.json example with version constraints  
 {  
 "name": "your-vendor/your-package",  
 "require": {  
 "php": "^8.1",  
 "illuminate/support": "^9.0|^10.0"  
 },  
 "require-dev": {  
 "orchestra/testbench": "^7.0|^8.0",  
 "phpunit/phpunit": "^9.5|^10.0"  
 }  
 }  
   
 // GitHub workflow for testing multiple Laravel versions  
 name: Tests  
   
 on: [push, pull\_request]  
   
 jobs:  
 test:  
 runs-on: ubuntu-latest  
 strategy:  
 fail-fast: false  
 matrix:  
 php: [8.1, 8.2, 8.3]  
 laravel: [9.\*, 10.\*]  
 dependency-version: [prefer-lowest, prefer-stable]  
 include:  
 - laravel: 10.\*  
 testbench: 8.\*  
 - laravel: 9.\*  
 testbench: 7.\*  
 exclude:  
 - laravel: 10.\*  
 php: 8.0  
   
 name: P${{ matrix.php }} - L${{ matrix.laravel }} - ${{ matrix.dependency-version }}  
   
 steps:  
 - name: Checkout code  
 uses: actions/checkout@v3  
   
 - name: Setup PHP  
 uses: shivammathur/setup-php@v2  
 with:  
 php-version: ${{ matrix.php }}  
 extensions: dom, curl, libxml, mbstring, zip, pcntl, pdo, sqlite, pdo\_sqlite, bcmath  
 coverage: xdebug  
   
 - name: Install dependencies  
 run: |  
 composer require "laravel/framework:${{ matrix.laravel }}" "orchestra/testbench:${{ matrix.testbench }}" --no-interaction --no-update  
 composer update --${{ matrix.dependency-version }} --prefer-dist --no-interaction  
   
 - name: Execute tests  
 run: vendor/bin/phpunit --coverage-clover=coverage.xml

* **CI/CD for packages**

# GitHub Actions workflow for release automation  
 name: Release  
   
 on:  
 push:  
 tags:  
 - "v\*"  
   
 jobs:  
 tests:  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v3  
 - uses: shivammathur/setup-php@v2  
 with:  
 php-version: 8.2  
 coverage: xdebug  
 - run: composer install --prefer-dist --no-interaction  
 - run: vendor/bin/phpunit  
   
 build-docs:  
 needs: tests  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v3  
 - name: Build API docs  
 run: |  
 composer require phpdocumentor/phpdocumentor  
 vendor/bin/phpdoc -d src/ -t docs/api/  
 - name: Deploy docs  
 uses: peaceiris/actions-gh-pages@v3  
 with:  
 github\_token: ${{ secrets.GITHUB\_TOKEN }}  
 publish\_dir: ./docs  
   
 release:  
 needs: [tests, build-docs]  
 runs-on: ubuntu-latest  
 steps:  
 - uses: actions/checkout@v3  
 with:  
 fetch-depth: 0  
   
 - name: Build Changelog  
 id: github\_release  
 uses: metcalfc/changelog-generator@v4.0.1  
 with:  
 myToken: ${{ secrets.GITHUB\_TOKEN }}  
   
 - name: Create GitHub Release  
 uses: actions/create-release@v1  
 env:  
 GITHUB\_TOKEN: ${{ secrets.GITHUB\_TOKEN }}  
 with:  
 tag\_name: ${{ github.ref }}  
 release\_name: Release ${{ github.ref }}  
 body: |  
 ${{ steps.github\_release.outputs.changelog }}  
 draft: false  
 prerelease: false

# Package Distribution & Maintenance

* **Publishing to Packagist**

// composer.json with package metadata  
 {  
 "name": "your-vendor/your-package",  
 "description": "A package for doing amazing things in Laravel",  
 "type": "library",  
 "license": "MIT",  
 "keywords": ["laravel", "package", "example"],  
 "homepage": "https://github.com/your-vendor/your-package",  
 "readme": "README.md",  
 "time": "2025-04-03",  
 "support": {  
 "issues": "https://github.com/your-vendor/your-package/issues",  
 "source": "https://github.com/your-vendor/your-package",  
 "docs": "https://your-vendor.github.io/your-package"  
 },  
 "authors": [  
 {  
 "name": "Your Name",  
 "email": "your.email@example.com",  
 "role": "Developer"  
 }  
 ],  
 "funding": [  
 {  
 "type": "github",  
 "url": "https://github.com/sponsors/your-name"  
 },  
 {  
 "type": "other",  
 "url": "https://buymeacoffee.com/your-name"  
 }  
 ],  
 "extra": {  
 "laravel": {  
 "providers": [  
 "YourVendor\YourPackage\YourPackageServiceProvider"  
 ],  
 "aliases": {  
 "YourPackage": "YourVendor\YourPackage\Facades\YourPackage"  
 }  
 }  
 }  
 }

* **Versioning strategy**

// Deprecation example in code  
 class YourService  
 {  
 /\*\*  
 \* Process the given data.  
 \*  
 \* @param array $data The data to process  
 \* @return array  
 \*   
 \* @deprecated since version 2.3.0, use processData() instead.  
 \*/  
 public function process(array $data): array  
 {  
 // Log deprecation notice during development  
 if (app()->environment('local', 'development', 'testing')) {  
 trigger\_deprecation(  
 'your-vendor/your-package',  
 '2.3.0',  
 'The %s method is deprecated, use %s instead.',  
 \_\_METHOD\_\_,  
 'processData()'  
 );  
 }  
   
 return $this->processData($data);  
 }  
   
 /\*\*  
 \* Process the given data with new implementation.  
 \*  
 \* @param array $data The data to process  
 \* @return array  
 \*/  
 public function processData(array $data): array  
 {  
 // New implementation  
 return $data;  
 }  
 }  
   
 // Upgrade guide in documentation (UPGRADE.md)  
 # Upgrade Guide  
   
 ## Upgrading from 1.x to 2.0  
   
 ### Breaking Changes  
   
 #### Configuration File Changes  
   
 The configuration file format has changed. You will need to republish the configuration:

#### API Changes  
   
 \* `OldService` has been removed. Use `NewService` instead.  
 \* The `process()` method now requires an array instead of a string.  
 \* The `getResults()` method now returns a Collection instead of an array.

* **Documentation best practices**

# Your Package Name  
   
 [![Latest Version on Packagist](https://img.shields.io/packagist/v/your-vendor/your-package.svg)](https://packagist.org/packages/your-vendor/your-package)  
 [![Tests](https://github.com/your-vendor/your-package/actions/workflows/tests.yml/badge.svg)](https://github.com/your-vendor/your-package/actions/workflows/tests.yml)  
 [![Total Downloads](https://img.shields.io/packagist/dt/your-vendor/your-package.svg)](https://packagist.org/packages/your-vendor/your-package)  
 [![License](https://img.shields.io/packagist/l/your-vendor/your-package.svg)](https://github.com/your-vendor/your-package/blob/main/LICENSE.md)  
   
 A brief description of what your package does and what problems it solves.  
   
 ## Installation  
   
 You can install the package via composer:

You can publish the config file with:

## Usage

### Advanced Examples  
   
 Here's how to use the package for more complex scenarios...  
   
 ## Testing

## Changelog  
   
 Please see [CHANGELOG](CHANGELOG.md) for more information on what has changed recently.  
   
 ## Contributing  
   
 Please see [CONTRIBUTING](CONTRIBUTING.md) for details.  
   
 ## Credits  
   
 - [Your Name](https://github.com/your-username)  
 - [All Contributors](../../contributors)  
   
 ## License  
   
 The MIT License (MIT). Please see [License File](LICENSE.md) for more information.

* **Handling community contributions**

# Contributing  
   
 We love contributions from the community! This document outlines the process for contributing to this package.  
   
 ## Code of Conduct  
   
 This project and everyone participating in it is governed by our [Code of Conduct](CODE\_OF\_CONDUCT.md). By participating, you are expected to uphold this code.  
   
 ## How Can I Contribute?  
   
 ### Reporting Bugs  
   
 Before submitting a bug report:  
   
 - Check the issue tracker to avoid duplicates  
 - Gather information to help us reproduce the issue  
 - Use the issue template when submitting  
   
 ### Suggesting Enhancements  
   
 Enhancement suggestions are tracked as GitHub issues:  
   
 - Use the feature request template  
 - Provide a clear rationale for the feature  
 - Describe the desired behavior  
   
 ### Pull Requests  
   
 1. Fork the repository  
 2. Create a branch: `git checkout -b feature/my-feature`  
 3. Make your changes and add tests  
 4. Run tests: `composer test`  
 5. Submit your PR with a clear description  
   
 ## Development Workflow  
   
 1. Clone the repository  
 2. Install dependencies: `composer install`  
 3. Run tests: `composer test`  
 4. Make your changes  
 5. Add tests for your changes  
 6. Ensure all tests pass  
   
 ## Style Guidelines  
   
 This project uses PHP-CS-Fixer for code style. Run `composer format` before submitting.

* **Security considerations**

# Security Policy  
   
 ## Supported Versions  
   
 | Version | Supported |  
 | ------- | ------------------ |  
 | 2.x.x | :white\_check\_mark: |  
 | 1.x.x | :x: |  
   
 ## Reporting a Vulnerability  
   
 We take security seriously. If you discover a security vulnerability within this package, please email security@example.com instead of using the issue tracker.  
   
 All security vulnerabilities will be promptly addressed. When reporting, please provide:  
   
 1. A description of the vulnerability  
 2. Steps to reproduce the issue  
 3. Possible impact of the vulnerability  
 4. Any suggestions for remediation if you have them  
   
 ## Security Update Process  
   
 When we receive a security bug report, we will:  
   
 1. Confirm the vulnerability and determine affected versions  
 2. Fix the issue and prepare a patch release  
 3. Release the patch and publicly disclose the issue after a reasonable period  
   
 For critical vulnerabilities, we will prioritize the fix and may release out-of-band patches.  
   
 ## Best Practices  
   
 As users of this package, we recommend:  
   
 - Always use the latest version  
 - Configure proper access controls when implementing the package  
 - Keep all dependencies up to date  
 - Use Composer's security advisories: `composer audit`

# Advanced Package Techniques

* **Package configuration cascading**

namespace YourVendor\YourPackage;  
   
 class ConfigurationManager  
 {  
 protected $config;  
 protected $defaultConfig;  
 protected $runtimeOverrides = [];  
   
 public function \_\_construct(array $config, array $defaultConfig)  
 {  
 $this->defaultConfig = $defaultConfig;  
 $this->config = array\_replace\_recursive($defaultConfig, $config);  
 $this->validateConfig();  
 }  
   
 public function get($key = null, $default = null)  
 {  
 // Check runtime overrides first  
 if ($key !== null && isset($this->runtimeOverrides[$key])) {  
 return $this->runtimeOverrides[$key];  
 }  
   
 // Get from config with dot notation support  
 if ($key === null) {  
 return $this->config;  
 }  
   
 return data\_get($this->config, $key, $default);  
 }  
   
 public function set($key, $value)  
 {  
 // Set a runtime configuration override  
 $this->runtimeOverrides[$key] = $value;  
   
 return $this;  
 }  
   
 public function reset($key = null)  
 {  
 if ($key === null) {  
 // Reset all runtime overrides  
 $this->runtimeOverrides = [];  
 } else {  
 // Reset specific key  
 unset($this->runtimeOverrides[$key]);  
 }  
   
 return $this;  
 }  
   
 public function toArray()  
 {  
 // Merge runtime overrides with config  
 return array\_replace\_recursive($this->config, $this->runtimeOverrides);  
 }  
   
 protected function validateConfig()  
 {  
 // Validate required configuration options  
 $requiredKeys = ['api\_key', 'connection'];  
   
 foreach ($requiredKeys as $key) {  
 if (empty($this->config[$key])) {  
 throw new \InvalidArgumentException("Missing required configuration: {$key}");  
 }  
 }  
   
 // Validate specific values  
 $allowedModes = ['sync', 'async', 'queue'];  
   
 if (!in\_array($this->config['mode'], $allowedModes)) {  
 throw new \InvalidArgumentException(  
 "Invalid mode: {$this->config['mode']}. Allowed values: " . implode(', ', $allowedModes)  
 );  
 }  
 }  
 }

* **Package discovery & extension**

namespace YourVendor\YourPackage;  
   
 class ExtensionManager  
 {  
 protected $extensions = [];  
 protected $app;  
   
 public function \_\_construct($app)  
 {  
 $this->app = $app;  
 }  
   
 public function register($name, $extension)  
 {  
 $this->extensions[$name] = $extension;  
   
 // If extension is a class name, resolve it from the container  
 if (is\_string($extension) && class\_exists($extension)) {  
 $this->extensions[$name] = $this->app->make($extension);  
 }  
   
 return $this;  
 }  
   
 public function extend($name, $callback)  
 {  
 if (!isset($this->extensions[$name])) {  
 throw new \InvalidArgumentException("Extension [{$name}] not registered.");  
 }  
   
 $extension = $this->extensions[$name];  
   
 // Apply the callback to the extension  
 $callback($extension);  
   
 return $this;  
 }  
   
 public function all()  
 {  
 return $this->extensions;  
 }  
   
 public function get($name)  
 {  
 if (!isset($this->extensions[$name])) {  
 throw new \InvalidArgumentException("Extension [{$name}] not registered.");  
 }  
   
 return $this->extensions[$name];  
 }  
 }  
   
 // Using macros for extending functionality  
 namespace YourVendor\YourPackage;  
   
 use Illuminate\Support\Traits\Macroable;  
   
 class QueryBuilder  
 {  
 use Macroable;  
   
 // Core methods...  
 }  
   
 // Registering macros to extend functionality  
 QueryBuilder::macro('whereLike', function ($attributes, $searchTerm) {  
 return $this->where(function ($query) use ($attributes, $searchTerm) {  
 foreach ((array) $attributes as $attribute) {  
 $query->orWhere($attribute, 'LIKE', "%{$searchTerm}%");  
 }  
 });  
 });

* **Artisan command generators**

namespace YourVendor\YourPackage\Commands;  
   
 use Illuminate\Console\Command;  
 use Illuminate\Support\Str;  
 use Illuminate\Filesystem\Filesystem;  
   
 class GenerateServiceCommand extends Command  
 {  
 protected $signature = 'make:your-service {name} {--interface : Create an interface for this service}'  
 . ' {--test : Create a test for this service}'  
 . ' {--force : Overwrite existing files}'  
 . ' {--namespace= : The namespace for the service}';  
   
 protected $description = 'Generate a new service class with optional interface and test';  
   
 protected $files;  
   
 public function \_\_construct(Filesystem $files)  
 {  
 parent::\_\_construct();  
 $this->files = $files;  
 }  
   
 public function handle()  
 {  
 $name = $this->argument('name');  
 $namespace = $this->option('namespace') ?: 'App\\Services';  
 $createInterface = $this->option('interface');  
 $createTest = $this->option('test');  
 $force = $this->option('force');  
   
 // Generate the service class  
 $this->createService($name, $namespace, $createInterface, $force);  
   
 // Generate the interface if requested  
 if ($createInterface) {  
 $this->createInterface($name, $namespace, $force);  
 }  
   
 // Generate the test if requested  
 if ($createTest) {  
 $this->createTest($name, $namespace, $force);  
 }  
   
 $this->info(Str::studly($name) . ' service generated successfully!');  
   
 // Run post-generation hooks  
 $this->runPostGenerationHooks($name, $namespace);  
   
 return Command::SUCCESS;  
 }  
   
 protected function createService($name, $namespace, $hasInterface, $force)  
 {  
 $className = Str::studly($name) . 'Service';  
 $interfaceName = $hasInterface ? Str::studly($name) . 'ServiceInterface' : null;  
   
 $stubPath = $hasInterface   
 ? \_\_DIR\_\_ . '/../stubs/service-with-interface.stub'   
 : \_\_DIR\_\_ . '/../stubs/service.stub';  
   
 $stub = $this->files->get($stubPath);  
   
 $stub = str\_replace(  
 ['{{ namespace }}', '{{ class }}', '{{ interface }}'],  
 [$namespace, $className, $interfaceName],  
 $stub  
 );  
   
 $path = $this->getPath($namespace, $className);  
   
 if (!$force && $this->files->exists($path)) {  
 $this->error($className . ' already exists!');  
 return;  
 }  
   
 $this->makeDirectory($path);  
 $this->files->put($path, $stub);  
   
 $this->info($className . ' created successfully.');  
 }  
   
 // Other methods for creating interfaces, tests, etc.  
 }

* **Module systems**

namespace YourVendor\ModuleSystem;  
   
 use Illuminate\Support\Facades\File;  
 use Illuminate\Contracts\Container\Container;  
   
 class ModuleManager  
 {  
 protected $app;  
 protected $modules = [];  
 protected $booted = false;  
   
 public function \_\_construct(Container $app)  
 {  
 $this->app = $app;  
 }  
   
 public function register($path)  
 {  
 $moduleConfig = $this->loadModuleConfig($path);  
   
 if (!$moduleConfig) {  
 return false;  
 }  
   
 $name = $moduleConfig['name'];  
   
 // Check dependencies  
 if (isset($moduleConfig['depends']) && is\_array($moduleConfig['depends'])) {  
 foreach ($moduleConfig['depends'] as $dependency) {  
 if (!isset($this->modules[$dependency])) {  
 throw new \RuntimeException("Module {$name} depends on {$dependency} which is not loaded.");  
 }  
 }  
 }  
   
 // Register the module  
 $this->modules[$name] = [  
 'config' => $moduleConfig,  
 'path' => $path,  
 'active' => true,  
 ];  
   
 // Register service provider if exists  
 if (isset($moduleConfig['provider'])) {  
 $this->app->register($moduleConfig['provider']);  
 }  
   
 return true;  
 }  
   
 public function boot()  
 {  
 if ($this->booted) {  
 return;  
 }  
   
 foreach ($this->modules as $name => $module) {  
 if (!$module['active']) {  
 continue;  
 }  
   
 $this->bootModule($name, $module);  
 }  
   
 $this->booted = true;  
 }  
   
 protected function bootModule($name, $module)  
 {  
 $config = $module['config'];  
   
 // Load translations  
 if (is\_dir($module['path'] . '/resources/lang')) {  
 $this->app['translator']->addNamespace($name, $module['path'] . '/resources/lang');  
 }  
   
 // Load views  
 if (is\_dir($module['path'] . '/resources/views')) {  
 $this->app['view']->addNamespace($name, $module['path'] . '/resources/views');  
 }  
   
 // Load routes  
 if (isset($config['routes']) && is\_array($config['routes'])) {  
 foreach ($config['routes'] as $file) {  
 $path = $module['path'] . '/' . $file;  
 if (File::exists($path)) {  
 require $path;  
 }  
 }  
 }  
   
 // Fire module booted event  
 if (isset($config['listeners']['boot'])) {  
 $listener = $config['listeners']['boot'];  
 if (is\_callable($listener)) {  
 $listener($this->app);  
 }  
 }  
 }  
   
 protected function loadModuleConfig($path)  
 {  
 $configPath = $path . '/module.json';  
   
 if (!File::exists($configPath)) {  
 return null;  
 }  
   
 return json\_decode(File::get($configPath), true);  
 }  
 }

* **Integrating with Laravel ecosystem**

namespace YourVendor\YourPackage;  
   
 use Laravel\Nova\Nova;  
 use Laravel\Nova\Tool;  
 use Illuminate\Support\Facades\Event;  
 use Illuminate\Support\Facades\Gate;  
 use Laravel\Horizon\Horizon;  
   
 class YourPackageServiceProvider extends ServiceProvider  
 {  
 public function register()  
 {  
 // Register core services  
 $this->app->singleton('your-package', function ($app) {  
 return new YourPackage($app['config']['your-package']);  
 });  
 }  
   
 public function boot()  
 {  
 // Nova integration  
 $this->bootNova();  
   
 // Horizon integration  
 $this->bootHorizon();  
   
 // Telescope integration  
 $this->bootTelescope();  
   
 // Listen for events from other packages  
 $this->registerEventListeners();  
 }  
   
 protected function bootNova()  
 {  
 if (!class\_exists(Nova::class)) {  
 return;  
 }  
   
 Nova::serving(function () {  
 Nova::tools([  
 new class extends Tool {  
 public function boot()  
 {  
 Nova::script('your-package', \_\_DIR\_\_.'/../dist/js/tool.js');  
 Nova::style('your-package', \_\_DIR\_\_.'/../dist/css/tool.css');  
 }  
   
 public function menu(Request $request)  
 {  
 return MenuItem::make('Your Tool')  
 ->path('/your-package')  
 ->icon('chart-bar');  
 }  
 },  
 ]);  
   
 // Register resource  
 Nova::resources([  
 YourResource::class,  
 ]);  
 });  
 }  
   
 protected function bootHorizon()  
 {  
 if (!class\_exists(Horizon::class)) {  
 return;  
 }  
   
 // Add custom Horizon metrics  
 Horizon::auth(function ($request) {  
 return Gate::check('viewHorizon', [$request->user()]);  
 });  
   
 // Tag jobs from this package  
 Horizon::tag(function ($job) {  
 if ($job instanceof YourPackageJob) {  
 return ['your-package'];  
 }  
   
 return [];  
 });  
 }  
   
 protected function bootTelescope()  
 {  
 if (!class\_exists(\Laravel\Telescope\Telescope::class)) {  
 return;  
 }  
   
 // Configure Telescope watcher for package  
 \Laravel\Telescope\Telescope::tag(function ($entry) {  
 if ($entry->type === 'request' && Str::startsWith($entry->content['uri'], '/your-package')) {  
 return ['your-package'];  
 }  
   
 return [];  
 });  
 }  
   
 protected function registerEventListeners()  
 {  
 // Listen for events from other packages  
 Event::listen('other-package.event', function ($event) {  
 // React to the event  
 });  
   
 // Provide your own events for others  
 $this->app['events']->listen('your-package.\*', function ($eventName, array $data) {  
 // Log events for debugging  
 if (config('your-package.debug')) {  
 logger("Event fired: {$eventName}", $data);  
 }  
 });  
 }  
 }