Laravel Modular Project Structure

Organized by domain modules with services and repositories. Scales well for larger teams.

#laravel #php #web #backend #modular #services

PNG PDF Copy Prompt

Project Directory

```
myproject/
    artisan
    composer.json
    package.json
    .env
    .env.example
  app/
    Http/
      Controllers/ Thin, delegate …
        Controller.php
        Auth/
        User/
        Product/
      Requests/ Form request va…
        User/
        Product/
      Middleware/
      Resources/ API Resources
    Models/
      User.php
      Product.php
      Order.php
      Traits/ Reusable model …
        HasUuid.php
        Searchable.php
    Services/ Business logic …
      UserService.php
      ProductService.php
      OrderService.php
      PaymentService.php
    Repositories/ Data access abs…
      UserRepository.php
      ProductRepository.php
      Contracts/
        UserRepositoryInterface.php
    Actions/ Single-purpose …
      CreateOrder.php
      ProcessPayment.php
    DTOs/ Data transfer o…
      UserData.php
      OrderData.php
    Events/
      OrderPlaced.php
    Listeners/
      SendOrderConfirmation.php
    Jobs/ Queue jobs
      ProcessPayment.php
    Providers/
    Exceptions/
  bootstrap/
  config/
  database/
    migrations/
    seeders/
    factories/
  resources/
    views/
      layouts/
      components/
      users/
      products/
      css/
      js/
  routes/
    web.php
    api.php
  tests/
    Feature/
      UserTest.php
      ProductTest.php
    Unit/
      Services/
      Repositories/
  public/
  storage/
```

Why This Structure?

This structure separates your code into clear layers: Controllers handle HTTP, Services contain business logic, and Repositories manage data access. Each layer is independently testable. DTOs and Actions further improve code organization for complex domains.

Key Directories

app/Services/ - Business logic, injected into controllers

app/Repositories/ - Data access layer, abstracted from Eloquent

app/Actions/ - Single-purpose classes for complex operations

app/DTOs/ - Typed data containers between layers

app/Http/Requests/ - Validation logic moved out of controllers

Service Layer Example

```
// app/Services/UserService.php
class UserService
{
    public function __construct(
        private UserRepository $users
    ) {}

    public function create(UserData $data): User
    {
        return $this->users->create($data->toArray());
    }
}
```

Getting Started

1. `composer create-project laravel/laravel myproject`
2. Create `app/Services`, `app/Repositories`, `app/DTOs` directories
3. Register repository interfaces in `AppServiceProvider`
4. `php artisan make:provider RepositoryServiceProvider`
5. `php artisan serve`

When To Use This

- Apps with complex business logic
- Teams of 3+ developers
- Need for unit testing business logic
- Domain with clear bounded contexts
- Long-term projects requiring maintainability

Trade-offs

More abstractions - Service/Repository layers add indirection

Boilerplate - More files per feature (DTO, Service, Repository)

Overkill for CRUD - Simple apps don't need this complexity

Naming Conventions

Services - `UserService`, `OrderService` - noun + Service

Repositories - `UserRepository` + `UserRepositoryInterface`

Actions - Verb + Noun: `CreateOrder`, `ProcessPayment`

DTOs - `UserData`, `OrderData` - append Data suffix