Design Pattern In Laravel Applications

LUU THANH SANG | NGƯỜI VIẾT MÃ

LARAVEL LIVE HÀ NỘI 2023

nvmcommunity.io</>

NGƯỜI VIẾT MÃ

About

Lưu Thanh Sang

Works

Software Architect at Getfly

Activities

Community Leader at Người Viết Mã

Coaching at NVM Heavy Booster Program

Contacts

Telegram: @imcaptainbolt

What is design pattern?

Design Pattern

A software design pattern is a general, reusable solution to a commonly occurring problem within a given context in software design.

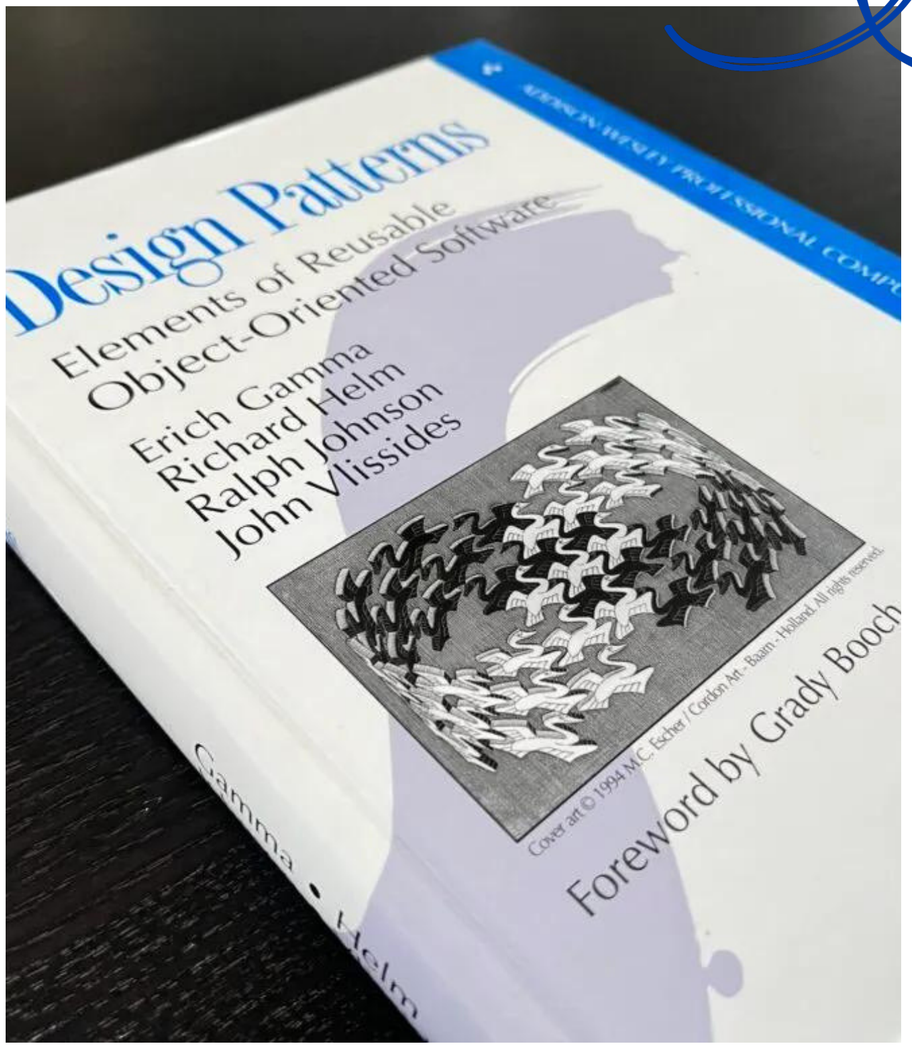
It is not a finished design that can be transformed directly into source or machine code. Rather, it is a description or template for how to solve a problem that can be used in many different situations.

– https://en.wikipedia.org/wiki/Design_Patterns

Books about design patterns

Design Patterns: Elements of Reusable Object-Oriented Software

Gangs of Four (GoF) Design Patterns



Gangs of Four (GoF) Design Patterns

Creational Design Patterns

- Factory Method
- Abstract Factory
- Singleton
- Prototype
- Builder

Structural Design Patterns

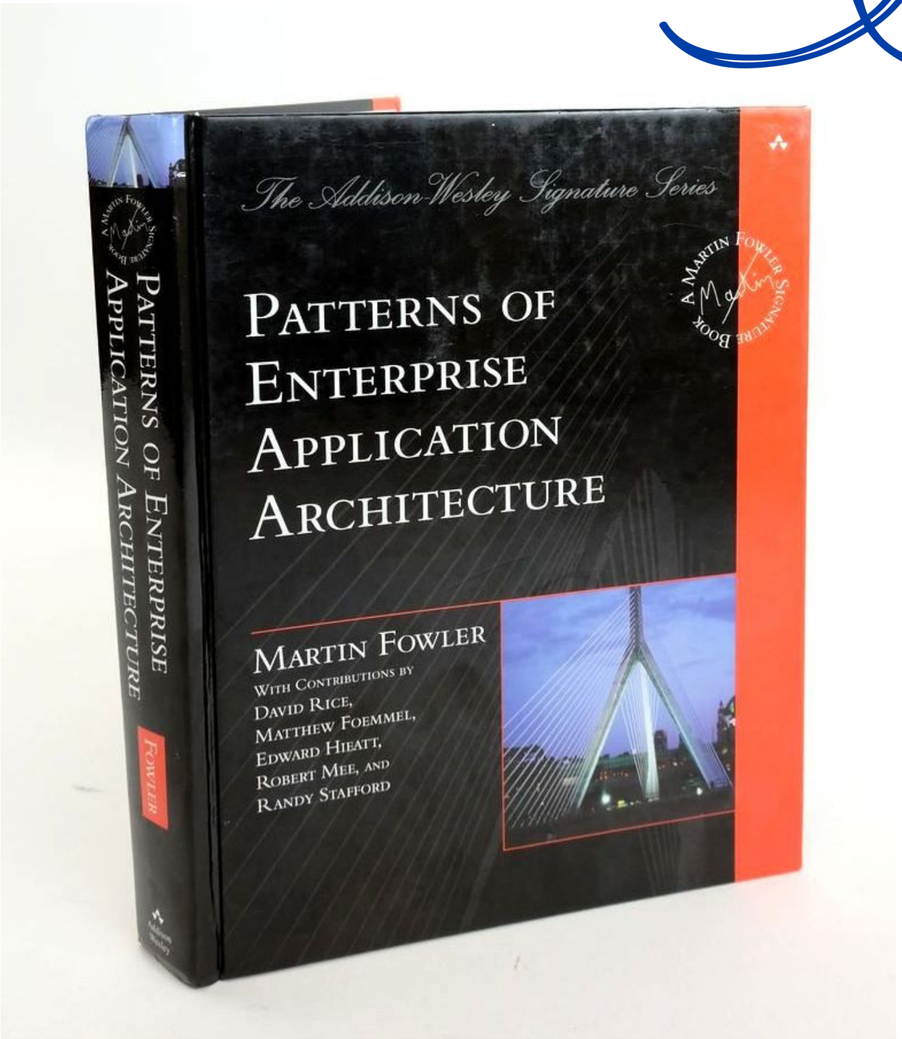
- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

Behavioral Design Patterns

- Template Method
- Mediator
- Chain of Responsibility
- Observer
- Strategy
- Command
- State
- Visitor
- Interpreter
- Iterator
- Memento

Patterns of Enterprise Application Architecture

P of EAA - Martin Fowler



P of EAA - Martin Fowler

Domain Logic Patterns

- Transaction Script
- Domain Model
- Table Module
- Service Layer

Data Source Architectural

- Table Data Gateway
- Row Data Gateway
- Active Record
- Data Mapper

Object-Relational Behavioral

- Unit of Work
- Identity Map
- Lazy Load

Object-Relational Structural

- Identity Field
- Foreign Key Mapping
- Association Table Mapping
- Dependent Mapping
- Embedded Value
- Serialized LOB
- Single Table Inheritance
- Class Table Inheritance
- Concrete Table Inheritance
- Inheritance Mappers

Web Presentation

- Model View Controller
- Page Controller
- Front Controller
- Template View
- Transform View
- Two-Step View
- Application Controller

Base

- Gateway
- Mapper
- Layer Supertype
- Separated Interface
- RegistryValue Object
- Money
- Special Case
- Plugin
- Service Stub
- Record Set

Distribution

- Remote Facade
- Data Transfer Object

Offline Concurrency

- Optimistic Offline Lock
- Pessimistic Offline Lock
- Coarse Grained Lock
- Implicit Lock

Object-Relational Metadata Mapping

- Metadata Mapping
- Query Object
- Repository

Session State

- Client Session State
- Server Session State
- Database Session State

Why should I learn patterns?

Design patterns are a toolkit of tried and tested solutions to common problems in software design.

Design patterns define a common language that you and your teammates can use to communicate more efficiently.

Laravel

The PHP Framework for Web Artisans

Laravel is a web application framework with expressive, elegant syntax. We've already laid the foundation — freeing you to create without sweating the small things.

Model-View-Controller (MVC)

Laravel follows the MVC pattern, which separates an application into three main components: Models (represent data and business logic), Views (handle user interfaces), and Controllers (manage requests and control the flow of the application).

Service Layer

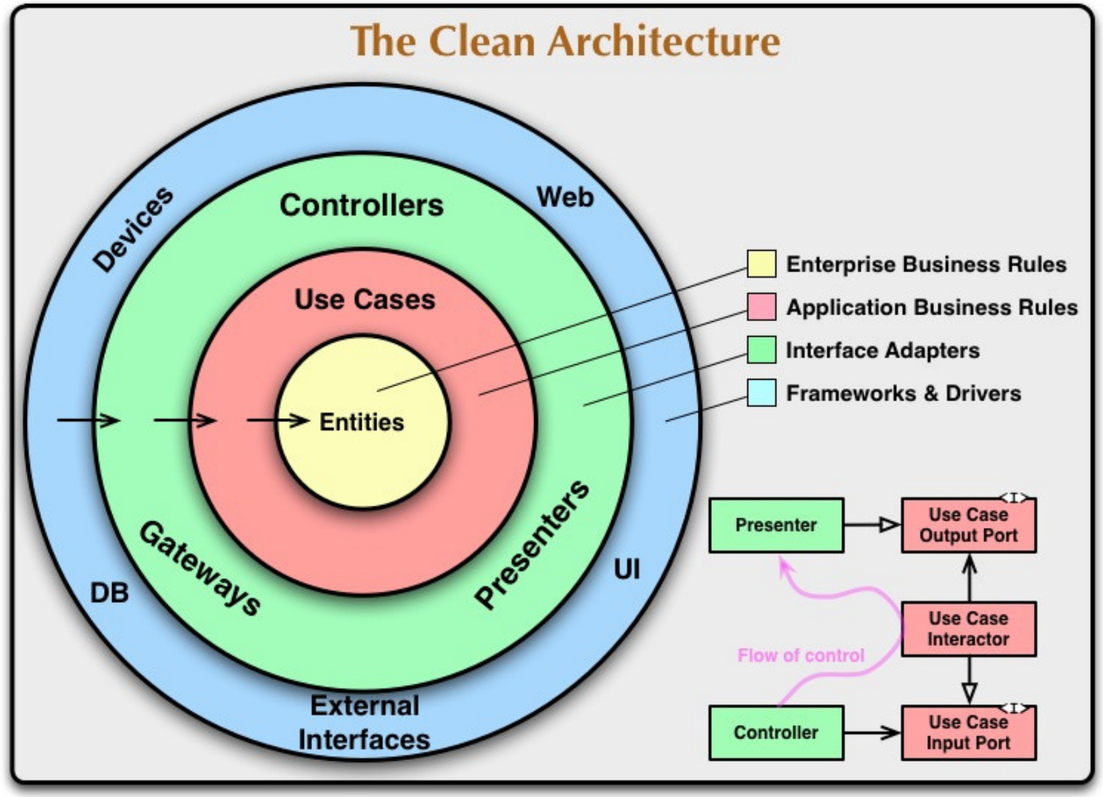
Defines an application's boundary with a layer of services that establishes a set of available operations and coordinates the application's response in each operation.

```php
namespace App\Http\Controllers;

use App\Services\UserService;
use App\Http\Requests\UserRequest;

class UserController
{
    protected $userService;

    public function __construct(UserService $userService)
    {
        $this->userService = $userService;
    }

    public function create(UserRequest $request)
    {
        $result = $this->userService->create($request->input());

        return back()->with($result);
    }
}
```

Clean Architecture

(Architectural pattern)

Clean Architecture is a software design approach that emphasizes independence and testability of components by **separating business logic, user interfaces, and technology concerns**, making applications more maintainable, extensible, and testable.



Eloquent

Laravel includes Eloquent, an object-relational mapper (ORM) that makes it enjoyable to interact with your database. When using Eloquent, **each database table has a corresponding "Model" that is used to interact with that table**. In addition to **retrieving records** from the database table, Eloquent models allow you to **insert**, **update**, and **delete records** from the table as well.

– https://laravel.com/docs/10.x/eloquent

```php
<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;

class Flight extends Model
{
    /**
     * The table associated with the model.
     *
     * @var string
     */
    protected $table = 'my_flights';
}

// client code

$flight = new Flight;

$flight->name = $request->name;

$flight->save();
```

Active Record

An object that **wraps a row in a database table or view**, encapsulates the **database access**, and **adds domain logic** on that data.

Eloquent x Repository?

Repository

Mediates between the domain and data mapping layers using a collection-like interface for **accessing domain objects**.

Eloquent x Repository?

Service Container

(Architecture Concepts)

The Laravel service container is a powerful tool for **managing class dependencies** and **performing dependency injection**. Dependency injection is a fancy phrase that essentially means this: class dependencies are "injected" into the class via the constructor or, in some cases, "setter" methods.

– https://laravel.com/docs/10.x/container

```php
<?php

namespace App\Http\Controllers;

use App\Http\Controllers\Controller;
use App\Repositories\UserRepository;
use App\Models\User;
use Illuminate\View\View;

class UserController extends Controller
{
    /**
     * Create a new controller instance.
     */
    public function __construct(
        protected UserRepository $users,
    ) {}

    /**
     * Show the profile for the given user.
     */
    public function show(string $id): View
    {
        $user = $this->users->find($id);

        return view('user.profile', ['user' => $user]);
    }
}
```

Dependency Injection

Dependency injection is a programming technique in which **an object or function receives other objects or functions that it depends on**

The 4 roles in Dependency Injection

- The **service** you want to use. (UserRepository)
- The **client** that uses the service. (UserController)
- An **interface** that's used by the client and implemented by the service. (UserRepositoryInterface)
- The **injector** which creates a service instance and injects it into the client. (ServiceContainer)

Dependency injection aims to separate the concerns of constructing objects and using them, leading to loosely coupled programs.

Singleton

Singleton is a creational design pattern that lets you **ensure that a class has only one instance**, while providing a global access point to this instance.

```
$this->app->singleton(UserRepositoryInterface::class, UserRepository::class);
```

Middleware

Middleware provide a convenient mechanism for inspecting and filtering HTTP requests entering your application.Middleware provide a convenient mechanism for inspecting and filtering HTTP requests entering your application.

```php
<?php

namespace App\Http\Middleware;

use Closure;
use Illuminate\Http\Request;
use Symfony\Component\HttpFoundation\Response;

class EnsureTokenIsValid
{
    /**
     * Handle an incoming request.
     *
     * @param  \Closure(\Illuminate\Http\Request): (\Symfony\Component\HttpFoundation\Response)  $next
     */
    public function handle(Request $request, Closure $next): Response
    {
        if ($request->input('token') !== 'my-secret-token') {
            return redirect('home');
        }

        return $next($request);
    }
}
```

Pipeline

A design pattern or architectural approach where **data or objects are passed through a sequence of processing stages or functions**, often represented as a chain of method or function calls. This pattern is used to simplify and modularize code, making it more readable and maintainable. **Each stage of the pipeline performs a specific operation on the data or object and passes the result to the next stage in a linear, sequential manner.**

Events

Laravel's events provide a **simple observer pattern implementation**, allowing you to subscribe and listen for various events that occur within your application

```php
use App\Events\OrderShipped;
use App\Listeners\SendShipmentNotification;

/**
 * The event listener mappings for the application.
 *
 * @var array<class-string, array<int, class-string>>
 */
protected $listen = [
    OrderShipped::class => [
        SendShipmentNotification::class,
    ],
];
```

Observer

Observer is a behavioral design pattern that lets you define a subscription mechanism to **notify multiple objects about any events that happen to the object they're observing.**

Understanding design patterns while learning Laravel simplifies code organization, promotes best practices, and enhances code reusability and maintainability.

Thank you for listening!

Q&A

Lưu Thanh Sang

Works

Software Architect at Getfly

Activities

Community Leader at Người Viết Mã

Coaching at NVM Heavy Booster Program

Contacts

Telegram: @imcaptainbolt