Bài 3
Laravel Controller

Module: BOOTCAMP WEB-BACKEND DEVELOPMENT WITH LARAVEL 2.1

Mục tiêu

- Trình bày được controller
- Trình bày được middleware
- Trình bày được cơ chế dependency injection
- Tạo được controller cơ bản
- Thao tác được với form trong ứng dụng Laravel
- Trình bày được kiến trúc Repository
- Triển khai được kiến trúc Repository

Controllers

Định nghĩa Controllers

Controllers và Namespaces

Controllers chỉ có một action

Controller

- Controller là các đối tượng được dùng để tiếp nhận các HTTP Request và trả về các HTTP Response
- Các lớp controller được đặt trong thư mục app/Http/Controllers
- Controller chứa các action là các phương thức xử lý các request
- Các action của controller nhận được request thông qua cơ chế điều hướng của route

Định nghĩa Controllers

- Thừa kế lớp Controller
- Có sẵn một số phương thức để sử dụng
- Ví dụ:

```
class UserController extends Controller {
    /**
    * Show the profile for the given user.
    */
    public function showProfile($id)
    {
        $user = User::findOrFail($id);
        return view('user.profile', ['user' => $user]);
    }
}
```

Lưu ý: Không bắt buộc phải kế thừa lớp Controller khi khai báo các controller, nhưng chúng ta sẽ không thể sử dụng được một số tính năng có sẵn hoặc một số phương thức như ‘middlware’, ‘validate’ hay ‘dispatch’.

Sử dụng Controller

- Mỗi controller có thể chứa nhiều phương thức (action) để xử lý các request khác nhau
- Có thể trỏ các route tới từng action của controller bằng cách chỉ định tên của controller và tên của phương thức tương ứng
- Ví dụ:

```
use App\Http\Controllers\UserController;

Route::get('/user/{id}', [UserController::class, 'show']);
```

  - Khi một request được gửi đến route ‘user/{id}’ thì nó sẽ được điều hướng đến phương thức show().

Controllers & Namespaces

- Nếu các controller được đặt trong thư mục con của thư mục App\Http\Controllers thì cần chỉ định tên của thư mục con đó
- Tên của thư mục con này cũng là một phần của namespace
- Trong route, cần chỉ rõ tên của namespace của controller
- Ví dụ, nếu lớp controller App\Http\Controllers\Photos\AdminController thì cần khai báo namespace trong route là Photos:

```
Route::get('foo’, [Photos\AdminController::class,’method’]);
```

Controller với chỉ một action

- Sử dụng phương thức __invoke trong controller để khai báo một controller với chỉ một action duy nhất, ví dụ:

```
class ShowProfile extends Controller
{
    public function __invoke($id)
    {
        return view('user.profile', ['user' => User::findOrFail($id)]);
    }
}
```

- Khi khai báo trong route thì không cần ghi tên của action
- Ví dụ:

```
Route::get('user/{id}', ShowProfile::class);
```

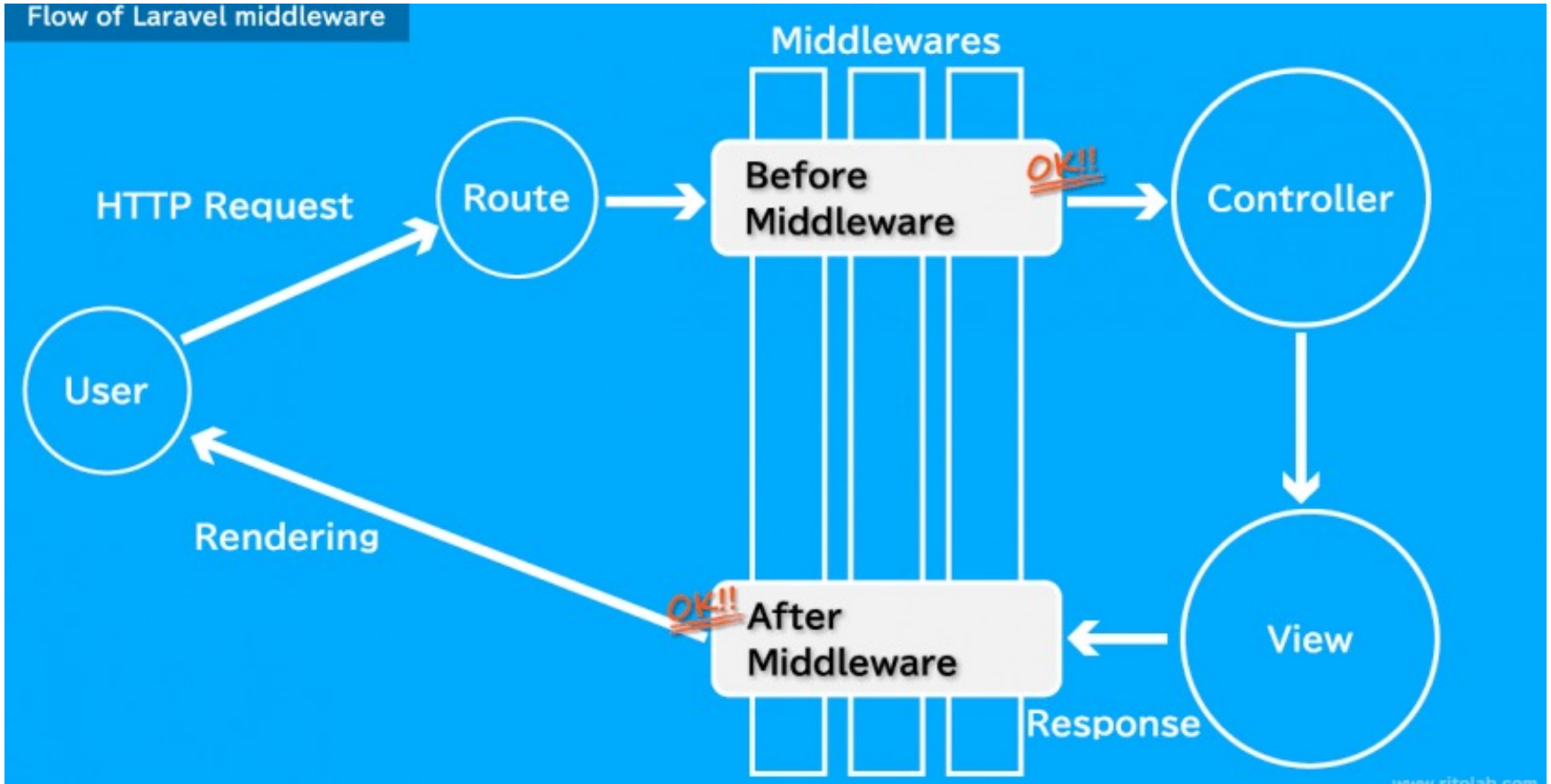
Middleware

Middleware

Đăng ký middleware trong controller

Middleware

- Middleware là cơ chế cho phép lọc các request trước khi chúng được tiếp nhận và xử lý bởi các controller hoặc là lọc các response trước khi chúng được gửi về cho client.



Định nghĩa middleware

- Sử dụng lệnh make:middleware để tạo middleware
- Ví dụ:

```
php artisan make:middleware CheckAge
```

  - Lớp CheckAge sẽ được tạo trong thư mục app/Http/Middleware

```php
namespace App\Http\Middleware;
use Closure;

class CheckAge
{
    public function handle($request, Closure $next)
    {
        //Code here
    }
}
```

Lọc request dựa trên điều kiện

- Ví dụ:
  - Chỉ cho phép truy cập nếu age >= 200
  - Nếu age < 200 thì điều hướng về trang ‘home’

```php
public function handle($request, Closure $next)
{
    if ($request->age <= 200) {
        return redirect('home');
    }
    return $next($request);
}
```

BeforeMiddleware

- Thực thi trước khi request được gửi đến controller
- Ví dụ:

```php
<?php
namespace App\Http\Middleware;

use Closure;

class BeforeMiddleware
{
    public function handle($request, Closure $next)
    {
        // Perform action

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

AfterMiddleware

- Thực thi sau khi request được xử lý bởi controller, trước khi response được trả về cho client
- Ví dụ:

```php
namespace App\Http\Middleware;

use Closure;

class AfterMiddleware
{
    public function handle($request, Closure $next)
    {
        $response = $next($request);
        // Perform action
        return $response;
    }
}
```

Đăng ký middleware cho controller (1)

- Có thể đăng ký middleware cho controller trong file route
- Ví dụ:

```
Route::get('profile', [UserController::class, 'show'])->middleware('auth');
```

Đăng ký middleware cho controller (2)

- Có thể đăng ký middleware bên trong constructor của controller
- Ví dụ:

```php
class UserController extends Controller
{
    public function __construct()
    {
        $this->middleware('auth');
        $this->middleware('log')->only('index');
        $this->middleware('subscribed')->except('store');
    }
}
```

Đăng ký middleware cho controller (3)

- Có thể định nghĩa middlware ngay bên trong constructor của controller thông qua một Closure
- Ví dụ:

```
$this->middleware(function ($request, $next) {
    // ...
    return $next($request);
});
```

Resource Controller

Resouce controller

- Laravel cung cấp một tiện ích cho phép nhanh chóng khởi tạo một controller với đầy đủ các action CRUD thông dụng
- Có thể đạt được dễ dàng bằng tham số --resource khi tạo controller
- Ví dụ:

```
php artisan make:controller PhotoController --resource
```

Lớp được tạo là app/Http/Controllers/PhotoController.php

- Cần đăng ký route trỏ tới resource controller.
- Ví dụ:

```php
use App\Http\Controllers\PhotoController;

Route::resource('photos', PhotoController::class);
```

Các phương thức của resource controller

Verb	URI	Action	Route Name
GET	`/photos`	index	photos.index
GET	`/photos/create`	create	photos.create
POST	`/photos`	store	photos.store
GET	`/photos/{photo}`	show	photos.show
GET	`/photos/{photo}/edit`	edit	photos.edit
PUT/PATCH	`/photos/{photo}`	update	photos.update
DELETE	`/photos/{photo}`	destroy	photos.destroy

Giả lập các method của form

- Trong HTML chỉ có thể khai báo các phương thức GET và POST của HTTP
- Không thể khai báo các phương thức như PUT hay DELETE
- Có thể giả lập các phương thức bằng cách thêm một trường có tên là '_method' vào trong form HTML
- Sử dụng directive '@method' để khai báo các phương thức giả lập
- Ví dụ:

```
<form action="/foo/bar" method="POST">
    @method('PUT')
</form>
```

Dependency Injection

Tiêm sự phụ thuộc

Dependency Injection

- Dependency Injection (tiêm phụ thuộc) là cơ chế để giúp dễ dàng sử dụng các đối tượng của một lớp bên trong một lớp khác
- Dependency Injection là cơ chế được sử dụng phổ biến trong nhiều các framework hiện nay, bao gồm cả Laravel
- Ví dụ, A phụ thuộc vào B:

```
class A {
    public function importantMethod() {
        $b = new B();
        $b->usefulMethod();
        //More code here
    }
}
```

- Có thể sử dụng Dependency Injection để tiêm đối tượng của B vào trong A

Dependency Injection: Ví dụ

```php
<?php
namespace App\Http\Controllers;

use App\Repositories\UserRepository;

class UserController extends Controller
{
    protected $users;

    public function __construct(UserRepository $users)
    {
        $this->users = $users;
    }
}
```

- Đối tượng của UserRepository được tự động tiêm vào UserController thông qua constructor

Tiêm phụ thuộc sử dụng method

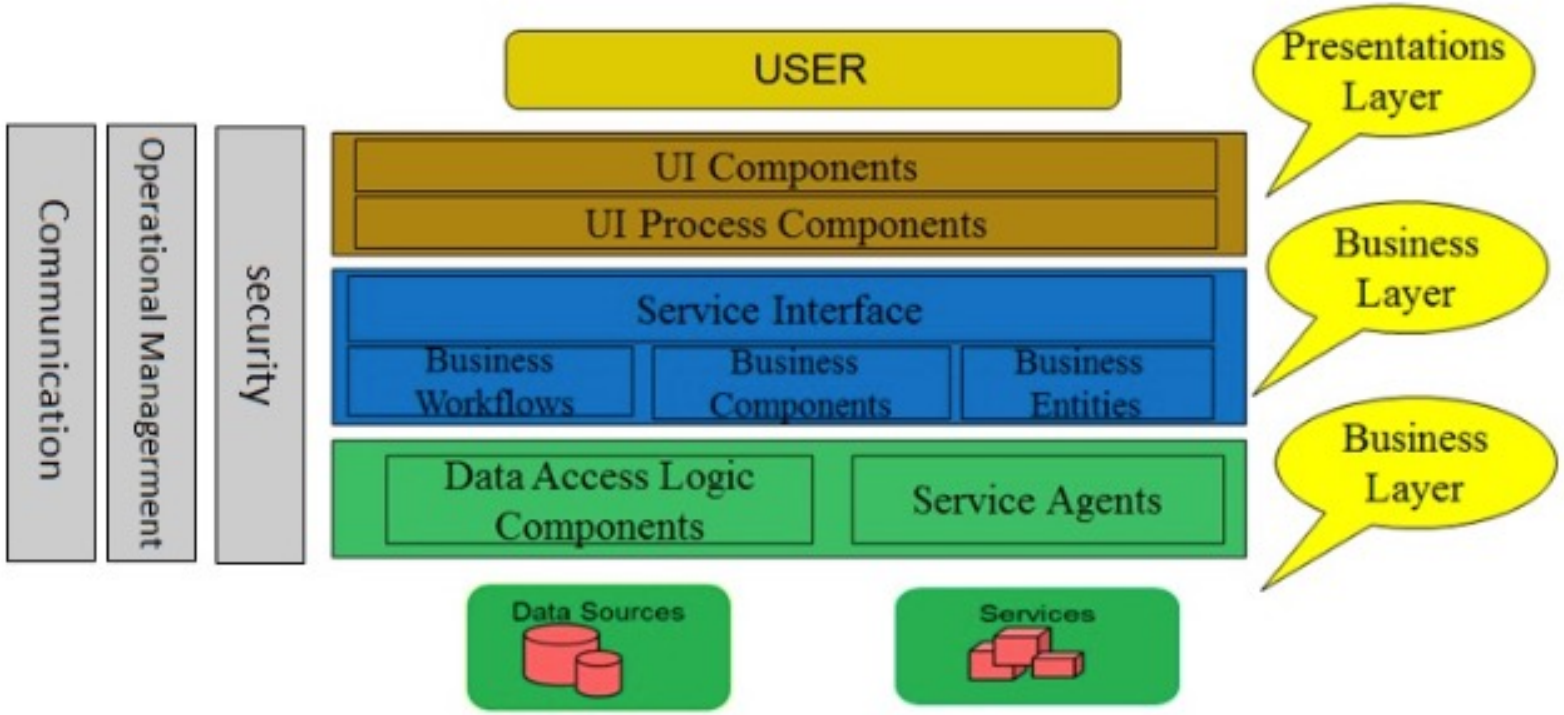
- Ngoài việc triển khai Dependency Injection thông qua constructor, có thể triển khai thông qua cơ chế gợi ý kiểu dữ liệu của tham số của các phương thức
- Ví dụ:

```
class UserController extends Controller
{
    public function store(Request $request)
    {
        $name = $request->name;
        //
    }
}
```

  - Đối tượng của Request được tự động tiêm vào UserController thông qua phương thức store

Repository Pattern

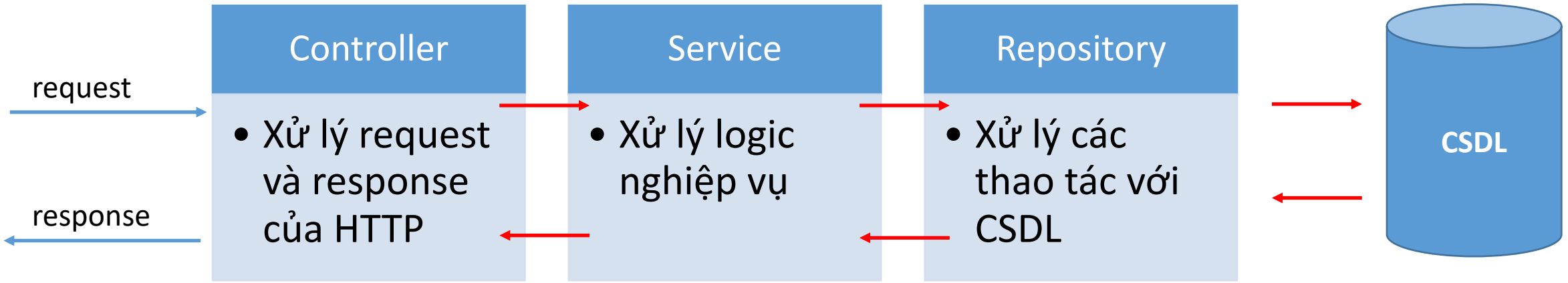
Mô hình 3 lớp (3 – layer)



Repository Pattern

- Là một mẫu kiến trúc cho phép tách biệt các tầng khác nhau của ứng dụng
- Là lớp trung gian giữa tầng Business Logic và Data Access, giúp cho việc truy cập dữ liệu chặt chẽ và bảo mật hơn.
- Giúp mã nguồn trở nên trong sáng, dễ duy trì và mở rộng hơn
- Các tầng trong repository pattern:
  - Tầng controller: Xử lý request và response của HTTP
  - Tầng service: Xử lý các logic nghiệp vụ
  - Tầng repository: Xử lý các thao tác truy xuất CSDL

Repository Pattern



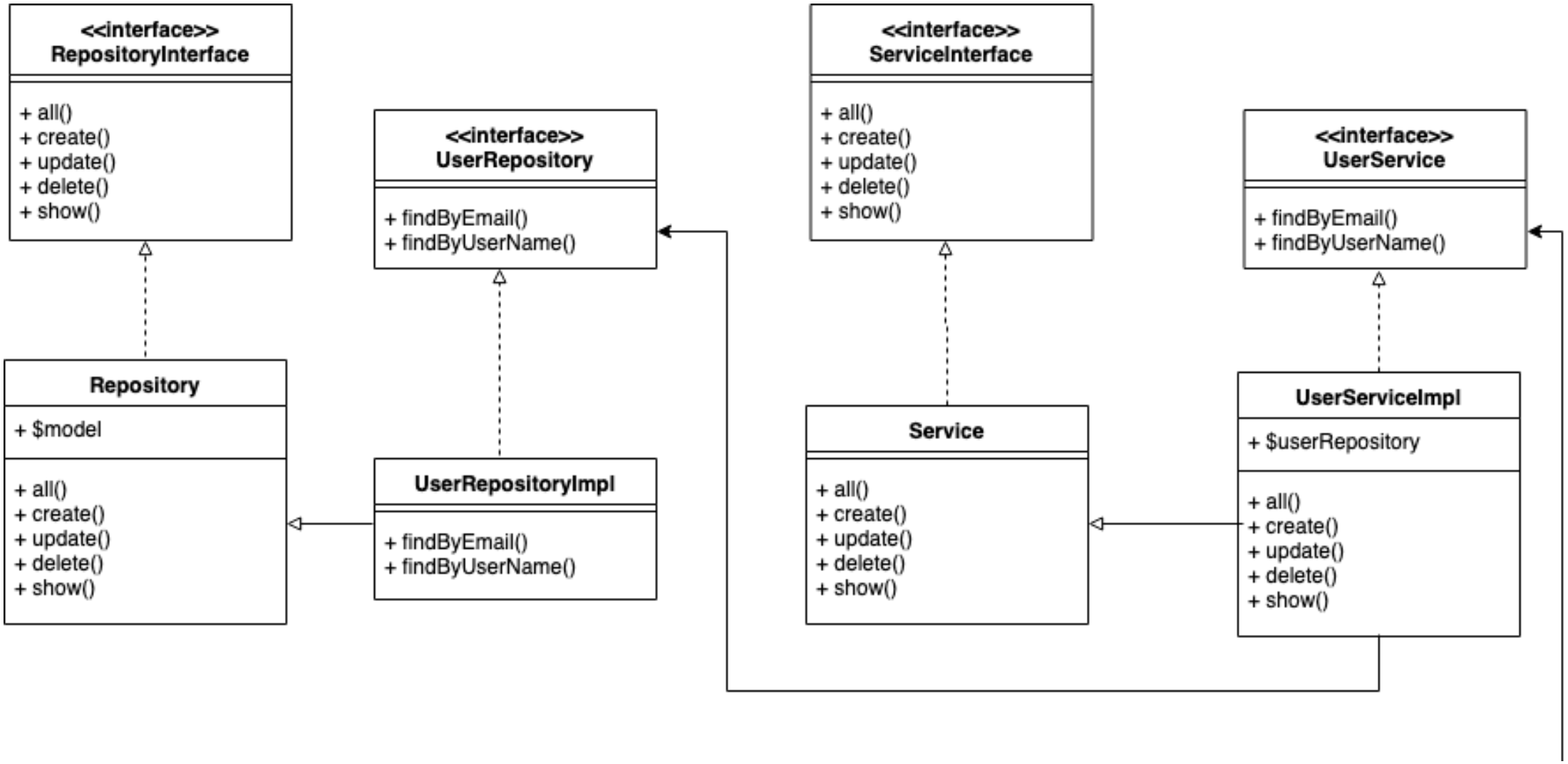
Lợi ích của Repository Pattern

- Mã nguồn gọn gàng, minh bạch
- Dễ truy xuất mã nguồn, dễ duy trì
- Dễ thay đổi
  - Khi thay đổi ở một tầng thì không ảnh hưởng đến các tầng còn lại
- Dễ mở rộng
  - Thông qua cơ chế kế thừa
  - Thông qua cơ chế dependency injection
- Dễ triển khai kiểm thử tự động
  - Có thể kiểm thử lần lượt cho từng tầng

Triển khai repository pattern

1. Xây dựng tầng repository
2. Xây dựng tầng service
   - Sử dụng cơ chế dependency injection để tiêm repository vào trong service
3. Xây dựng tầng controller
   - Sử dụng cơ chế dependency injection để tiêm service vào trong controller

Sơ đồ UML


<<interface>>
RepositoryInterface
+ all()
+ create()
+ update()
+ delete()
+ show()
<<interface>>
UserRepository
+ findByEmail()
+ findByUserName()
<<interface>>
ServiceInterface
+ all()
+ create()
+ update()
+ delete()
+ show()
<<interface>>
UserService
+ findByEmail()
+ findByUserName()
Repository
+ $model
+ all()
+ create()
+ update()
+ delete()
+ show()
UserRepositoryImpl
+ findByEmail()
+ findByUserName()
Service
+ all()
+ create()
+ update()
+ delete()
+ show()
UserServiceImpl
+ $userRepository
+ all()
+ create()
+ update()
+ delete()
+ show()

Tóm tắt bài học

- Controller là các đối tượng được dùng để tiếp nhận các HTTP Request và trả về các HTTP Response.
- Các action của controller nhận được request thông qua cơ chế điều hướng của route.
- Middleware là cơ chế cho phép lọc các request trước khi chúng được tiếp nhận và xử lý bởi các controller hoặc là lọc các response trước khi chúng được gửi về cho client.
- Dependency Injection (tiêm phụ thuộc) là cơ chế để giúp dễ dàng sử dụng các đối tượng của một lớp bên trong một lớp khác
- Việc áp dụng design pattern là rất cần thiết khi xây dựng phát triển ứng dụng.

Hướng dẫn

- Hướng dẫn làm bài thực hành và bài tập
- Chuẩn bị bài tiếp: ***Views & Blade***