

BÀI TẬP THỰC HÀNH

Buổi 05

Mục tiêu:

- Rèn luyện Migration và làm quen với Eloquent ORM: Thông qua việc tạo và quản lý cơ sở dữ liệu bằng migration, người thực hành sẽ hiểu được cách Laravel tương tác với cơ sở dữ liệu và cách sử dụng Eloquent ORM để thao tác dữ liệu.
- Thực hiện CRUD: Để hiểu rõ cách thức Laravel xử lý các thao tác cơ bản như Tạo (Create), Đọc (Read), Cập nhật (Update) và Xóa (Delete) dữ liệu.
- Integrate Bootstrap với Laravel: Làm quen với cách tích hợp Bootstrap vào Laravel, sử dụng Bootstrap để tạo ra một giao diện người dùng đẹp mắt và phản hồi.
- Làm việc với Blade Template Engine: Thực hành cách sử dụng Blade, template engine của Laravel, để hiển thị dữ liệu và tạo các components giao diện linh hoạt.
- Làm việc với quan hệ giữa các bảng: Thực hành việc tạo và quản lý quan hệ giữa các bảng trong cơ sở dữ liệu, cụ thể trong bài này là quan hệ giữa authors và books.

Lưu ý: Các nội dung được tham khảo từ ChatGPT nên có thể cần kiểm tra lại khi tham chiếu kiến thức. Kỹ thuật sử dụng ChatGPT đã hướng dẫn SV trên lớp một cách chi tiết để tăng tốc độ khi tìm hiểu kiến thức lập trình. SV không lạm dụng chỉ để giải quyết việc viết mã mà không thực sự tìm hiểu để hiểu kiến thức.

NỘI DUNG HƯỚNG DẪN

Bước 1: Khởi tạo dự án

```
composer create-project --prefer-dist laravel/laravel laravel-crud  
cd laravel-crud
```

Bước 2: Cài đặt Bootstrap

- Sinh viên có thể sử dụng Bootstrap CDN (nếu có kết nối mạng) hoặc tải về Bootstrap và sử dụng Offline hoặc cài đặt Bootstrap thông qua composer

Bước 3: Tạo cơ sở dữ liệu và cấu hình .env

- Tạo một cơ sở dữ liệu trên máy chủ MySQL hoặc MariaDB của bạn. Sau đó, cập nhật file

.env:

DB_DATABASE=tên_cơ_sở_dữ_liệu

DB_USERNAME=tên_người_dùng

DB_PASSWORD=mật_khẩu

Bước 4: Tạo hai bảng: authors và books (ví dụ)

- Tạo migrations:

```
php artisan make:migration create_authors_table
```

```
php artisan make:migration create_books_table
```

- Cập nhật các files migration:

Trong database/migrations/[timestamp]_create_authors_table.php:

```
public function up()
{
    Schema::create('authors', function (Blueprint $table) {
        $table->id();
        $table->string('name');
        $table->timestamps();
    });
}
```

Trong database/migrations/[timestamp]_create_books_table.php:

```
public function up()
{
    Schema::create('books', function (Blueprint $table) {
        $table->id();
        $table->unsignedBigInteger('author_id');
        $table->foreign('author_id')->references('id')->on('authors');
        $table->string('title');
        $table->timestamps();
    });
}
```

- Áp dụng migrations:

```
php artisan migrate
```

Bước 5: Sử dụng Faker để thêm dữ liệu Fake

Để thêm dữ liệu giả vào cơ sở dữ liệu, chúng ta có thể sử dụng thư viện Faker được tích hợp sẵn trong Laravel. Laravel đã tích hợp sẵn Faker trong composer.json nên bạn không cần phải cài đặt thêm.

- Tạo Seeder:

```
php artisan make:seeder AuthorsTableSeeder
```

```
php artisan make:seeder BooksTableSeeder
```

- Chỉnh sửa AuthorsTableSeeder: Mở file database/seeder/AuthorsTableSeeder.php và thêm dữ liệu giả:

```
use Illuminate\Database\Seeder;
use Faker\Factory as Faker;
use Illuminate\Support\Facades\DB;
use App\Models\Book;
use App\Models\Author;
class AuthorsTableSeeder extends Seeder
{
    public function run()
    {
        DB::table('authors')->delete();

        $faker = Faker::create();

        for ($i = 0; $i < 50; $i++) {
            Author::create([
                'name' => $faker->name,
            ]);
        }
    }
}
```

- Chỉnh sửa BooksTableSeeder: Mở file database/seeder/BooksTableSeeder.php và thêm dữ liệu giả:

```
use Illuminate\Database\Seeder;
use Faker\Factory as Faker;
```

```

use Illuminate\Support\Facades\DB;
use App\Models\Book;
use App\Models\Author;

class BooksTableSeeder extends Seeder
{
    public function run()
    {
        DB::table('books')->delete();
        $faker = Faker::create();
        $author_ids = Author::all()->pluck('id')->toArray();
        for ($i = 0; $i < 100; $i++) {
            Book::create([
                'author_id' => $faker->randomElement($author_ids),
                'title' => $faker->sentence(5),
            ]);
        }
    }
}

```

- Thực hiện seeding dữ liệu:

Cách 1: Thông qua DatabaseSeeder.php

Mở file database/seeder/DatabaseSeeder.php:

```

public function run()
{
    $this->call([
        AuthorsTableSeeder::class,
        BooksTableSeeder::class,
    ]);
}

```

php artisan db:seed

Cách 2: Thông qua tham số dòng lệnh

php artisan db:seed --class=AuthorsTableSeeder


php artisan db:seed --class=BooksTableSeeder

Bước 6: Tạo Models và Controllers đồng thời (chú ý tham số -mcr)

php artisan make:model Author -mcr

php artisan make:model Book -mcr

Trong lệnh php artisan make:model Author -mcr, các tham số -mcr có ý nghĩa sau:

-m hoặc --migration: Khi thêm tham số này, Laravel sẽ tự động tạo một migration mới liên quan đến model bạn tạo. Migration này sẽ giúp bạn xây dựng cấu trúc bảng dữ liệu cho model.  **Bỏ qua -m do/nếu Bạn đã tạo migration độc lập ở trước**

-c hoặc --controller: Thêm tham số này sẽ yêu cầu Laravel tạo một controller mới cho model. Controller này mặc định sẽ không có bất kỳ phương thức nào.

-r hoặc --resource: Khi sử dụng cùng với -c, tham số này sẽ yêu cầu Laravel tạo một controller tài nguyên (resource controller) với tất cả các phương thức cần thiết cho CRUD (tức là: index, create, store, show, edit, update, destroy).

Vì vậy, lệnh php artisan make:model Author -mcr sẽ tạo ra:

Một model tên là Author.

Một migration cho bảng authors (theo quy ước, tên bảng sẽ là phiên bản số nhiều của tên model).

Một resource controller cho model Author với tất cả các phương thức CRUD.

Bước 7: Định nghĩa routes

Trong routes/web.php:

```
Route::resource('authors', AuthorController::class);
```

```
Route::resource('books', BookController::class);
```

Bước 8: Xây dựng giao diện và cập nhật controller để xử lý CRUD

Lúc này, bạn đã có tất cả các Models, Controllers, và Routes. Bạn cần phát triển các phương thức trong mỗi Controller (index, create, store, show, edit, update, destroy) để xử lý CRUD cho cả Author và Book.

Dùng Bootstrap 5 trong các views Blade để làm cho giao diện trở nên đẹp mắt hơn. Sử dụng các components như navbar, tables, forms, etc.

❖ Xử lý chức năng R – Read

"R" trong CRUD đại diện cho "Read", nghĩa là đọc hoặc lấy thông tin. Trong ngữ cảnh web, điều này thường liên quan đến việc lấy dữ liệu từ cơ sở dữ liệu và hiển thị nó lên giao diện trang web. Trong Laravel, chúng ta sẽ thực hiện "R" thông qua một số bước cơ bản sau đây:

1. Controller:

Trong resource controller, phương thức chính liên quan đến "Read" là **index** và **show**.

index: Dùng để lấy ra danh sách tất cả các bản ghi.

show: Dùng để lấy chi tiết của một bản ghi cụ thể dựa trên ID.

Ví dụ:

```
public function index()
{
    $authors = Author::all();
    return view('authors.index', compact('authors'));
}

public function show(Author $author)
{
    return view('authors.show', compact('author'));
}
```

2. Eloquent ORM:

Laravel sử dụng Eloquent, một Object-Relational Mapping (ORM) tool, để thao tác với cơ sở dữ liệu.

all(): Lấy tất cả bản ghi.

find(\$id): Tìm bản ghi dựa trên ID.

where(): Lọc dữ liệu dựa trên điều kiện.

3. Blade Views:

Sử dụng Blade template engine để hiển thị dữ liệu đã lấy từ cơ sở dữ liệu.

Ví dụ cho `index.blade.php`:

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

Ví dụ cho `show.blade.php`:

```
<h1>{{ $author->name }}</h1>
<p>{{ $author->bio }}</p>
```

4. Routes:

Cần định nghĩa routes trong `routes/web.php` để xử lý việc hiển thị.

```
Route::resource('authors', AuthorController::class);
```

Khi sử dụng `Route::resource()`, Laravel sẽ tự động xử lý các routes liên quan đến CRUD, bao gồm cả "R". Ví dụ:

`/authors` sẽ điều hướng đến index method (danh sách tất cả authors).

`/authors/{author}` sẽ điều hướng đến show method (hiển thị chi tiết của một author).

Như vậy, "R" trong CRUD đại diện cho việc đọc/lấy thông tin từ cơ sở dữ liệu và hiển thị nó. Trong Laravel, chúng ta sử dụng Eloquent để truy vấn cơ sở dữ liệu, Blade để hiển thị dữ liệu, và Routing để định nghĩa cách dữ liệu được truy cập và hiển thị trên web.

❖ Xử lý chức năng D – Delete

Xử lý "D" trong CRUD đại diện cho việc "Delete". Để thêm một lớp bảo mật và trải nghiệm người dùng tốt hơn, chúng ta có thể sử dụng một Modal (cửa sổ bật lên) để xác nhận trước khi xóa một mục.

Dưới đây là hướng dẫn chi tiết về việc thực hiện chức năng xóa với Modal để xác nhận:

1. Controller:

Trong controller, chúng ta cần một phương thức **destroy** để xử lý việc xóa.

```
public function destroy(Author $author)
{
    $author->delete();
    return redirect()->route('authors.index')->with('success', 'Author
```

```
deleted successfully');  
}
```

2. Blade View:

Trong view, thêm nút xóa và một modal Bootstrap:

```
<!-- Button trigger modal -->  
<button type="button" class="btn btn-danger" data-bs-toggle="modal"  
data-bs-target="#deleteModal-{{ $author->id }}">  
    Delete  
</button>  
  
<!-- Modal -->  
<div class="modal fade" id="deleteModal-{{ $author->id }}" tabindex="-  
1" aria-labelledby="exampleModalLabel" aria-hidden="true">  
    <div class="modal-dialog">  
        <div class="modal-content">  
            <div class="modal-header">  
                <h5 class="modal-title">Delete Confirmation</h5>  
                <button type="button" class="btn-close" data-bs-dismiss="modal"  
aria-label="Close"></button>  
            </div>  
            <div class="modal-body">  
                Are you sure you want to delete this author?  
            </div>  
            <div class="modal-footer">  
                <button type="button" class="btn btn-secondary" data-bs-  
dismiss="modal">Close</button>  
                <form action="{{ route('authors.destroy', $author->id) }}"  
method="POST">  
                    @csrf  
                    @method('DELETE')  
                    <button type="submit" class="btn btn-danger">Delete</button>  
                </form>  
            </div>
```



```
</div>
</div>
</div>
```

3. Routes:

Đảm bảo bạn đã định nghĩa route tương ứng trong routes/web.php:

```
Route::resource('authors', AuthorController::class);
```

Khi sử dụng Route::resource(), Laravel sẽ tự động xử lý route cho phương thức destroy.

4. JavaScript (Bootstrap):

Để modal hoạt động, đảm bảo bạn đã kết nối thư viện Bootstrap JavaScript. Thêm liên kết sau vào phần dưới cùng của file Blade (hoặc trong layouts/app.blade.php nếu bạn sử dụng layout):

```
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta3/dist/js/bootstrap.bundle.min.js"></script>
```

Với các bước trên, khi bạn nhấn nút "**Delete**", một modal sẽ xuất hiện yêu cầu xác nhận trước khi xóa. Chỉ khi người dùng nhấn "**Delete**" trong modal, bản ghi sẽ thực sự bị xóa khỏi cơ sở dữ liệu.

❖ Xử lý chức năng C – Create

"C" trong CRUD đại diện cho "Create", nghĩa là việc thêm một bản ghi mới vào cơ sở dữ liệu. Dưới đây là hướng dẫn chi tiết về việc thực hiện chức năng tạo mới trong Laravel:

1. Controller:

Trong controller, chúng ta sẽ cần hai phương thức: **create** (để hiển thị form nhập dữ liệu) và **store** (để xử lý dữ liệu gửi từ form).

Ví dụ trong AuthorController.php:

```
public function create()
{
    return view('authors.create');
}

public function store(Request $request)
{
    $request->validate([
        'name' => 'required',
```

```

        'bio' => 'required',
    ]);

    Author::create($request->all());
    return redirect()->route('authors.index')
        ->with('success', 'Author created successfully.');
```

2. Blade View:

Tạo một file Blade cho form nhập dữ liệu, ví dụ [create.blade.php](#).

```

<form action="{{ route('authors.store') }}" method="POST">
    @csrf

    <div class="form-group">
        <label for="name">Name:</label>
        <input type="text" name="name" class="form-control" required>
    </div>

    <div class="form-group">
        <label for="bio">Bio:</label>
        <textarea name="bio" class="form-control" required></textarea>
    </div>

    <div class="form-group">
        <button type="submit" class="btn btn-primary">Submit</button>
    </div>
</form>
```

3. Routes:

Đảm bảo bạn đã định nghĩa các routes tương ứng trong routes/web.php.

```
Route::resource('authors', AuthorController::class);
```

Khi sử dụng Route::resource(), Laravel sẽ tự động xử lý các routes cho phương thức create và store.

4. Link tới trang tạo mới:

Trên trang danh sách (thường là [index.blade.php](#)), thêm một nút hoặc liên kết để người dùng

có thể truy cập trang tạo mới.

```
<a href="{{ route('authors.create') }}" class="btn btn-success">Add New Author</a>
```

Bằng cách thực hiện các bước trên, chúng ta đã xây dựng chức năng tạo mới trong Laravel. Khi người dùng nhấn nút "**Add New Author**", họ sẽ được chuyển đến trang nhập dữ liệu. Sau khi điền thông tin và gửi form, dữ liệu sẽ được lưu vào cơ sở dữ liệu và người dùng sẽ được chuyển hướng về trang danh sách với một thông báo thành công.

❖ Xử lý chức năng U – Update

"U" trong CRUD đại diện cho "Update", nghĩa là cập nhật một bản ghi hiện có trong cơ sở dữ liệu. Dưới đây là hướng dẫn chi tiết về việc thực hiện chức năng cập nhật trong Laravel:

1. Controller:

Trong controller, chúng ta sẽ cần hai phương thức: **edit** (để hiển thị form cập nhật) và **update** (để xử lý dữ liệu gửi từ form cập nhật).

Ví dụ trong AuthorController.php:

```
public function edit(Author $author)
{
    return view('authors.edit', compact('author'));
}

public function update(Request $request, Author $author)
{
    $request->validate([
        'name' => 'required',
        'bio' => 'required',
    ]);

    $author->update($request->all());
    return redirect()->route('authors.index')
        ->with('success', 'Author updated successfully.');
```

2. Blade View:

Tạo một file Blade cho form cập nhật, ví dụ edit.blade.php.

```

<form action="{{ route('authors.update', $author->id) }}" method="POST">
    @csrf
    @method('PUT')

    <div class="form-group">
        <label for="name">Name:</label>
        <input type="text" name="name" class="form-control" value="{{
$author->name }}" required>
    </div>

    <div class="form-group">
        <label for="bio">Bio:</label>
        <textarea name="bio" class="form-control" required>{{ $author-
>bio }}</textarea>
    </div>

    <div class="form-group">
        <button type="submit" class="btn btn-primary">Update</button>
    </div>
</form>

```

3. Routes:

Đảm bảo bạn đã định nghĩa các routes tương ứng trong [routes/web.php](#).

```
Route::resource('authors', AuthorController::class);
```

Khi sử dụng `Route::resource()`, Laravel sẽ tự động xử lý các routes cho phương thức edit và update.

4. Link tới trang cập nhật:

Trên trang danh sách (thường là [index.blade.php](#)) hoặc trang chi tiết ([show.blade.php](#)), thêm một nút hoặc liên kết để người dùng có thể truy cập trang cập nhật.

```

<a href="{{ route('authors.edit', $author->id) }}" class="btn btn-
primary">Edit</a>

```

Với việc thực hiện các bước trên, chúng ta đã xây dựng chức năng cập nhật cho ứng dụng Laravel. Người dùng có thể nhấn vào nút "**Edit**", sau đó họ sẽ được chuyển đến trang cập nhật với form đã điền sẵn dữ liệu hiện tại. Khi họ thay đổi thông tin và gửi form, dữ liệu sẽ được cập

nhật trong cơ sở dữ liệu và người dùng sẽ được chuyển hướng về trang danh sách với một thông báo thành công.

KẾT

Bài thực hành trên chỉ là một hướng dẫn cơ bản về cách xây dựng ứng dụng CRUD với Laravel 8 và Bootstrap 5. Trong thực tế, bạn cần thêm nhiều chi tiết và tính năng khác như validation, flash messages, pagination, etc. để hoàn thiện ứng dụng của mình.

PHỤ LỤC 1: Minh họa chức năng kế thừa giao diện trong Laravel Blade Template

Kế thừa giao diện là một trong những tính năng mạnh mẽ của hệ thống template Blade trong Laravel. Điều này cho phép bạn định nghĩa một layout chung và sau đó kế thừa hoặc mở rộng nó trong các views khác.

Giả sử chúng ta muốn xây dựng một trang web với header, footer và một vùng nội dung chính. Chúng ta có thể bắt đầu bằng cách tạo một layout chính.

1. Tạo một layout chính

File: <resources/views/layouts/app.blade.php>

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@yield('title', 'Default Title')</title>
    <link href="{{ asset('css/bootstrap.css') }}" rel="stylesheet">
</head>
<body>
    <header>
        <!-- Header content -->
    </header>

    <main>
        @yield('content')
    </main>

    <footer>
        <!-- Footer content -->
    </footer>

    <script src="{{ asset('js/bootstrap.js') }}"></script>
</body>
</html>
```

2. Tạo một view kế thừa từ layout chính

File: [resources/views/welcome.blade.php](#)

```
@extends('layouts.app')

@section('title', 'Welcome Page')

@section('content')
    <div class="container">
        <h1>Welcome to our website!</h1>
        <p>This is the main content area.</p>
    </div>
@endsection
```

Trong ví dụ trên:

`@extends('layouts.app')` chỉ định rằng view này sẽ kế thừa từ layout `app.blade.php`.

`@section('title', 'Welcome Page')` đặt tiêu đề cho trang. Nếu không có tiêu đề được đặt, thì tiêu đề mặc định (trong trường hợp này là "Default Title") sẽ được sử dụng.

`@section('content')` ... `@endsection` định nghĩa nội dung cho vùng `@yield('content')` trong layout chính.

Với kế thừa giao diện, bạn chỉ cần thay đổi layout chính một lần và tất cả các views kế thừa từ nó sẽ

PHỤ LỤC 2: Danh sách các kiểu dữ liệu khi tạo migrations

Trong Laravel, khi tạo migrations, bạn có thể sử dụng nhiều kiểu dữ liệu cho các cột trong bảng cơ sở dữ liệu. Dưới đây là một tóm tắt của một số kiểu dữ liệu phổ biến và mô tả ngắn gọn về chúng:

1. `$table->bigIncrements('id')`: Kiểu BIGINT tự tăng cho khóa chính.
2. `$table->bigInteger('votes')`: Kiểu BIGINT.
3. `$table->binary('data')`: Kiểu BLOB.
4. `$table->boolean('confirmed')`: Kiểu BOOLEAN.
5. `$table->char('name', 100)`: Kiểu CHAR với độ dài tối đa là 100.
6. `$table->date('created_at')`: Kiểu DATE.
7. `$table->dateTime('created_at', 0)`: Kiểu DATETIME với 0 số chữ số phần thập phân.
8. `$table->decimal('amount', 8, 2)`: Kiểu DECIMAL với 8 chữ số và 2 chữ số sau dấu phẩy.
9. `$table->double('column', 8, 2)`: Kiểu DOUBLE với độ dài và số chữ số sau dấu phẩy.
10. `$table->enum('choices', ['foo', 'bar'])`: Kiểu ENUM.
11. `$table->float('amount', 8, 2)`: Kiểu FLOAT với độ dài và số chữ số sau dấu phẩy.
12. `$table->geometry('positions')`: Kiểu GEOMETRY.
13. `$table->increments('id')`: Kiểu INTEGER tự tăng cho khóa chính.
14. `$table->integer('votes')`: Kiểu INTEGER.
15. `$table->ipAddress('visitor')`: Kiểu IP address.
16. `$table->json('options')`: Kiểu JSON.
17. `$table->jsonb('options')`: Kiểu JSONB (chủ yếu sử dụng cho cơ sở dữ liệu PostgreSQL).
18. `$table->longText('notes')`: Kiểu LONGTEXT.
19. `$table->macAddress('device')`: Kiểu MAC address.
20. `$table->mediumIncrements('id')`: Kiểu MEDIUMINT tự tăng cho khóa chính.
21. `$table->mediumInteger('numbers')`: Kiểu MEDIUMINT.
22. `$table->mediumText('description')`: Kiểu MEDIUMTEXT.
23. `$table->morphs('taggable')`: Thêm kiểu UNSIGNED BIGINT `taggable_id` và STRING `taggable_type`.
24. `$table->nullableMorphs('taggable')`: Tương tự như morphs nhưng cho phép NULL.
25. `$table->nullableTimestamps()`: Tương tự timestamps nhưng cho phép NULL.
26. `$table->rememberToken()`: Thêm một cột `remember_token` kiểu VARCHAR(100) có thể null.
27. `$table->smallIncrements('id')`: Kiểu SMALLINT tự tăng cho khóa chính.
28. `$table->smallInteger('votes')`: Kiểu SMALLINT.
29. `$table->softDeletes()`: Thêm một cột `deleted_at` kiểu TIMESTAMP có thể null để sử dụng với tính năng Soft Deleting.
30. `$table->string('name', 100)`: Kiểu VARCHAR với độ dài tối đa là 100.
31. `$table->text('description')`: Kiểu TEXT.
32. `$table->time('sunrise', 0)`: Kiểu TIME.
33. `$table->timestamp('added_on')`: Kiểu TIMESTAMP.
34. `$table->timestamps()`: Thêm hai cột `created_at` và `updated_at` kiểu TIMESTAMP.

Đây chỉ là một phần nhỏ trong số các kiểu dữ liệu mà bạn có thể sử dụng khi tạo migrations trong Laravel. Đọc tài liệu chính thức của Laravel sẽ cung cấp một cái nhìn đầy đủ và chi tiết hơn về mỗi kiểu dữ liệu và các tùy chọn đi kèm.

PHỤ LỤC 3: Danh sách các hàm phổ biến trong thư viện Faker

Thư viện Faker là một thư viện PHP giúp tạo dữ liệu giả mạo cho các mục đích như kiểm thử hoặc điền dữ liệu vào cơ sở dữ liệu mẫu. Dưới đây là một tóm tắt của một số hàm phổ biến trong thư viện Faker:

1. Cơ bản

- `name()`: Trả về một tên người giả mạo.
- `address()`: Trả về một địa chỉ giả mạo.
- `city()`: Trả về tên một thành phố giả mạo.
- `country()`: Trả về tên một quốc gia giả mạo.

2. Dành cho Text

- `sentence($nbWords = 6, $variableNbWords = true)`: Trả về một câu giả mạo.
- `paragraph($nbSentences = 3, $variableNbSentences = true)`: Trả về một đoạn văn giả mạo.

3. Dành cho Số điện thoại

- `phoneNumber()`: Trả về một số điện thoại giả mạo.

4. Dành cho Ngày & Giờ

- `date($format = 'Y-m-d', $max = 'now')`: Trả về một ngày giả mạo.
- `time($format = 'H:i:s', $max = 'now')`: Trả về một thời gian giả mạo.

5. Dành cho Internet

- `email()`: Trả về một địa chỉ email giả mạo.
- `userName()`: Trả về một tên người dùng giả mạo.
- `url()`: Trả về một URL giả mạo.

6. Dành cho Tập & Hình ảnh

- `image($dir = '/tmp', $width = 640, $height = 480)`: Trả về đường dẫn của một hình ảnh giả mạo.

7. Dành cho Kinh doanh

- `company()`: Trả về tên một công ty giả mạo.

8. Dành cho Sách

- `title()`: Trả về tiêu đề của một cuốn sách giả mạo.

9. Dành cho Số

- `randomNumber($nbDigits = null, $strict = false)`: Trả về một số ngẫu nhiên.

10. Dành cho UUID

- `uuid()`: Trả về một UUID giả mạo.

11. Dành cho Địa điểm

- `longitude()`: Trả về kinh độ giả mạo.
- `latitude()`: Trả về vĩ độ giả mạo.

Và còn nhiều hàm khác nữa. Thư viện Faker cung cấp một loạt các hàm để tạo ra hầu như mọi loại dữ liệu giả mạo bạn cần. Để biết danh sách đầy đủ và chi tiết hơn về các hàm trong thư viện Faker, bạn nên tham khảo tài liệu chính thức của thư viện này.

PHỤ LỤC 4: Tóm tắt ngôn ngữ truy vấn Eloquent ORM

Eloquent ORM là một phần quan trọng của Laravel, giúp làm việc với cơ sở dữ liệu trở nên dễ dàng và trực quan hơn. Dưới đây là tóm tắt về cách sử dụng ngôn ngữ truy vấn Eloquent dựa trên ví dụ CRUD ở trên:

Tạo (Create)

- Tạo một bản ghi mới:

```
$author = new Author;
```

```
$author->name = 'Tên tác giả';
```

```
$author->save();
```

- Hoặc sử dụng phương thức create:

```
$author = Author::create(['name' => 'Tên tác giả']);
```

Đọc (Read)

- Lấy tất cả bản ghi:

```
$authors = Author::all();
```

- Tìm bản ghi theo ID:

```
$author = Author::find(1);
```

- Sử dụng điều kiện truy vấn:

```
$authors = Author::where('name', 'like', '%nguyen%')->get();
```

- Lấy bản ghi đầu tiên phù hợp với điều kiện:

```
$author = Author::where('name', 'like', '%nguyen%')->first();
```

Cập nhật (Update)

- Cập nhật bản ghi:

```
$author = Author::find(1);
```

```
$author->name = 'Tên mới';
```

```
$author->save();
```

- Cập nhật bằng phương thức update:

```
Author::where('id', 1)->update(['name' => 'Tên mới']);
```

Xóa (Delete)

- Xóa một bản ghi:

```
$author = Author::find(1);
```

```
$author->delete();
```

- Xóa bản ghi dựa trên điều kiện:

```
Author::where('name', 'like', '%nguyen%')->delete();
```

Quan hệ

Với ví dụ ở trên, giả sử Author có nhiều Book:

- Lấy tất cả sách của một tác giả:

```
$books = $author->books;
```

- Thêm sách cho một tác giả:

```
$book = new Book(['title' => 'Tiêu đề sách']);
```

```
$author->books()->save($book);
```

Đây chỉ là một phần nhỏ về cách sử dụng Eloquent ORM trong Laravel. Eloquent cung cấp nhiều tính năng và phương thức khác nữa giúp truy vấn và tương tác với cơ sở dữ liệu một cách dễ dàng và linh hoạt.

PHỤ LỤC 5: Tóm tắt cách truy vấn theo kiểu Query Builder

Query Builder trong Laravel cung cấp một giao diện trực quan và linh hoạt để xây dựng và thực thi các truy vấn SQL. Dưới đây là tóm tắt về cách sử dụng Query Builder dựa trên ví dụ CRUD ở trên:

Tạo (Create)

- Thêm một bản ghi mới:

```
DB::table('authors')->insert([  
    'name' => 'Tên tác giả'  
]);
```

Đọc (Read)

- Lấy tất cả bản ghi:

```
$authors = DB::table('authors')->get();
```

- Lấy bản ghi theo ID:

```
$author = DB::table('authors')->where('id', 1)->first();
```

- Sử dụng điều kiện truy vấn:

```
$authors = DB::table('authors')->where('name', 'like', '%nguyen%')->get();
```

Cập nhật (Update)

- Cập nhật bản ghi:

```
DB::table('authors')->where('id', 1)->update(['name' => 'Tên mới']);
```

Xóa (Delete)

- Xóa một bản ghi:

```
DB::table('authors')->where('id', 1)->delete();
```

- Xóa dựa trên điều kiện:

```
DB::table('authors')->where('name', 'like', '%nguyen%')->delete();
```

Joins và Quan hệ

Giả sử bạn muốn kết nối bảng authors và books:

- Lấy tất cả sách của một tác giả:

```
$books = DB::table('books')
```

```
->join('authors', 'books.author_id', '=', 'authors.id')
```

```
->where('authors.id', 1)
```

```
->select('books.*')
```

```
->get();
```

Đây chỉ là một tóm tắt cơ bản về cách sử dụng Query Builder trong Laravel. Query Builder cung cấp nhiều tính năng và phương thức khác nữa giúp truy vấn và tương tác với cơ sở dữ liệu một cách dễ dàng và linh hoạt.

PHỤ LỤC 6: Tóm tắt truy vấn theo kiểu Raw Data

Sử dụng truy vấn Raw Data trong Laravel cho phép bạn thực thi truy vấn SQL trực tiếp mà không cần thông qua Eloquent ORM hoặc Query Builder. Dưới đây là tóm tắt về cách sử dụng truy vấn Raw Data dựa trên ví dụ CRUD ở trên:

Tạo (Create)

- Thêm một bản ghi mới:

```
DB::statement('INSERT INTO authors (name) VALUES (?)', ['Tên tác giả']);
```

Đọc (Read)

- Lấy tất cả bản ghi:

```
$authors = DB::select('SELECT * FROM authors');
```

- Lấy bản ghi theo ID:

```
$author = DB::select('SELECT * FROM authors WHERE id = ?', [1]);
```

- Sử dụng điều kiện truy vấn:

```
$authors = DB::select('SELECT * FROM authors WHERE name LIKE ?', ['%nguyen%']);
```

Cập nhật (Update)

- Cập nhật bản ghi:

```
DB::update('UPDATE authors SET name = ? WHERE id = ?', ['Tên mới', 1]);
```

Xóa (Delete)

- Xóa một bản ghi:

```
DB::delete('DELETE FROM authors WHERE id = ?', [1]);
```

- Xóa dựa trên điều kiện:

```
DB::delete('DELETE FROM authors WHERE name LIKE ?', ['%nguyen%']);
```

Joins và Quan hệ

Giả sử bạn muốn kết nối bảng authors và books:

- Lấy tất cả sách của một tác giả:

```
$books = DB::select('
    SELECT books.*
    FROM books
    JOIN authors ON books.author_id = authors.id
    WHERE authors.id = ?
    ', [1]);
```

Sử dụng truy vấn Raw Data cần cẩn trọng để tránh các vấn đề như SQL Injection. Laravel đã

cung cấp các biện pháp bảo vệ thông qua việc sử dụng bindings (như ví dụ ở trên) để giảm thiểu nguy cơ này. Tuy nhiên, khi có thể, bạn nên sử dụng Eloquent ORM hoặc Query Builder vì chúng cung cấp một giao diện trực quan hơn và có tính năng bảo mật cao hơn.