CHƯƠNG 6: GIỚI THIỆU VỀ LARAVEL

1. Lý thuyết Cốt lõi (Khái niệm)

- Framework là gì? Laravel là một PHP Framework. Hãy tưởng tượng nó là một "bộ khung" xe đã được lắp ráp sẵn (bánh xe, động cơ, khung sườn). Thay vì tự xây dựng mọi thứ từ đầu (như PHT 2-5), bạn chỉ cần tập trung vào việc "thêm" các tính năng (ghế da, màu sơn) cho chiếc xe. Nó cung cấp các quy tắc và công cụ để làm việc nhanh hơn và an toàn hơn.
- Composer: Là công cụ quản lý thư viện (dependency manager) cho PHP. Nó giúp bạn cài đặt Laravel và các thư viện khác (ví dụ: thư viện xử lý thanh toán) chỉ bằng một dòng lệnh.
- Artisan: Là "con dao đa năng" của Laravel. Đây là công cụ dòng lệnh (command-line) của Laravel. Thay vì tạo file SinhVienModel.php bằng tay, bạn chỉ cần gõ php artisan make:model SinhVien.
- Routing (Định tuyến) (6.5): Đây là "bảng chỉ dẫn" của ứng dụng.
    - Nó nằm trong file routes/web.php. o Nó định nghĩa: "Khi người dùng truy cập URL /gioi-thieu thì phải chạy code nào?".
- Controller (Điều khiển) (6.5): Đây là "bộ não" xử lý (giống hệt Controller ở Chương 5).
    - Nó nằm trong thư mục app/Http/Controllers/.
    - Đây là nơi chứa code logic (ví dụ: function showGioiThieu() { ... }).

Luồng hoạt động mới:

Người dùng truy cập URL $\rightarrow$ Laravel kiểm tra routes/web.php $\rightarrow$ Route gọi đến TenController@tenPhuongThuc $\rightarrow$ Controller xử lý logic và trả về kết quả.

2. Nhiệm vụ Thực hành (BẮT BUỘC)

Kịch bản: Cài đặt Laravel, tạo dự án đầu tiên, và xây dựng luồng "Route $\rightarrow$ Controller" cơ bản.

Code Khởi đầu (Starter Code):

Không có code, vì chúng ta sẽ dùng công cụ dòng lệnh (Terminal/CMD/GitBash).

---

A. Cài đặt (6.2, 6.3)

1. // TODO 1: Cài đặt Composer (Nếu bạn chưa cài): Truy cập getcomposer.org và cài đặt nó.
2. // TODO 2: Mở Terminal (CMD hoặc Git Bash), cd vào thư mục htdocs của XAMPP.
3. // TODO 3: Chạy lệnh sau để tạo dự án Laravel (đặt tên là cse485_chapter6). Lệnh này sẽ tự động tải về hàng trăm file cần thiết: Bash composer create-project --prefer-dist laravel/laravel cse485_chapter6
4. // TODO 4: Di chuyển vào thư mục dự án vừa tạo: Bash cd cse485_chapter6

5. // TODO 5: Khởi động server "nhúng" của Laravel (không cần XAMPP Apache): Bash

php artisan serve

(Nếu chạy thành công, nó sẽ báo server đang chạy ở http://127.0.0.1:8000)

---

B. Xây dựng Route và Controller (6.4, 6.5)

Mở một Terminal/CMD thứ hai (giữ nguyên Terminal ở TODO 5) và cd vào thư mục cse485_chapter6.

1. // TODO 6: Dùng Artisan để tạo một Controller mới tên là PageController: Bash

php artisan make:controller PageController

2. // TODO 7: Mở dự án bằng VSCode. Mở tệp Controller vừa tạo tại:

app/Http/Controllers/PageController.php

3. // TODO 8: Bên trong class PageController, thêm một phương thức (method) mới tên là showHomepage():

PHP

```
<?php namespace App\Http\Controllers;
use Illuminate\Http\Request;
class PageController extends Controller
{
// TODO 8: Thêm phương thức này public function
showHomepage()
{
// TODO 9: Thay vì echo, chúng ta 'return' return "Chào mừng bạn đến
với PHT Chương 6 - Laravel!";
}
}
```

4. // TODO 10: Mở tệp "Routing" tại routes/web.php.

5. // TODO 11: Import PageController của bạn ở đầu tệp:

PHP use App\Http\Controllers\PageController;

6. // TODO 12: Xóa Route Route::get('/', ...); mặc định. Thay vào đó, thêm 2 Route mới:

- Một Route cho URL / (trang chủ).
- Một Route cho URL /about.
- Cả hai đều trỏ đến PageController@showHomepage (chúng ta sẽ dùng chung 1 hàm cho PHT này).

PHP

```
// ... (sau dòng use)


// TODO 12: Thêm 2 route này
Route::get('/', [PageController::class, 'showHomepage']);
Route::get('/about', [PageController::class, 'showHomepage']);
```

3. Yêu cầu Bằng chứng (Proof of Work) Bạn phải nộp lại 3 bằng chứng sau: A. Code đã hoàn thiện:

1. Dán (paste) toàn bộ code của tệp app/Http/Controllers/PageController.php.
2. Dán (paste) toàn bộ code của tệp routes/web.php. B. Ảnh chụp màn hình Kết

quả (3 ẢNH):

1. Ảnh 1 (Terminal): Chụp màn hình Terminal sau khi chạy lệnh php artisan make:controller PageController thành công.
2. Ảnh 2 (Terminal): Chụp màn hình Terminal đang chạy lệnh php artisan serve.
3. Ảnh 3 (Trình duyệt Web): Chụp ảnh màn hình trình duyệt truy cập vào http://127.0.0.1:8000/ (phải thấy thông điệp chào mừng bạn đã return ở TODO 9). (Dán Code A1, A2 và 3 Ảnh B1, B2, B3 của bạn vào đây)

A1

```
<?php


namespace App\Http\Controllers;


use Illuminate\Http\Request;


class PageController extends Controller
{
   public function showHomepage()
   {
      return "Chào mừng bạn đến với PHT Chương 6 - Laravel!";
   }
```

}

A2

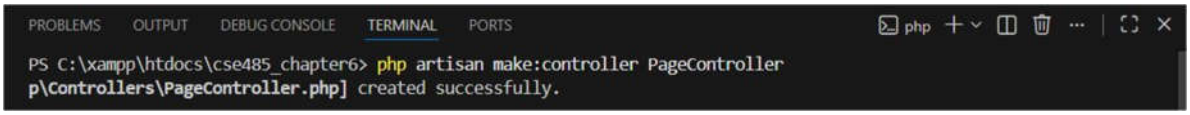
<?php

use Illuminate\Support\Facades\Route; use

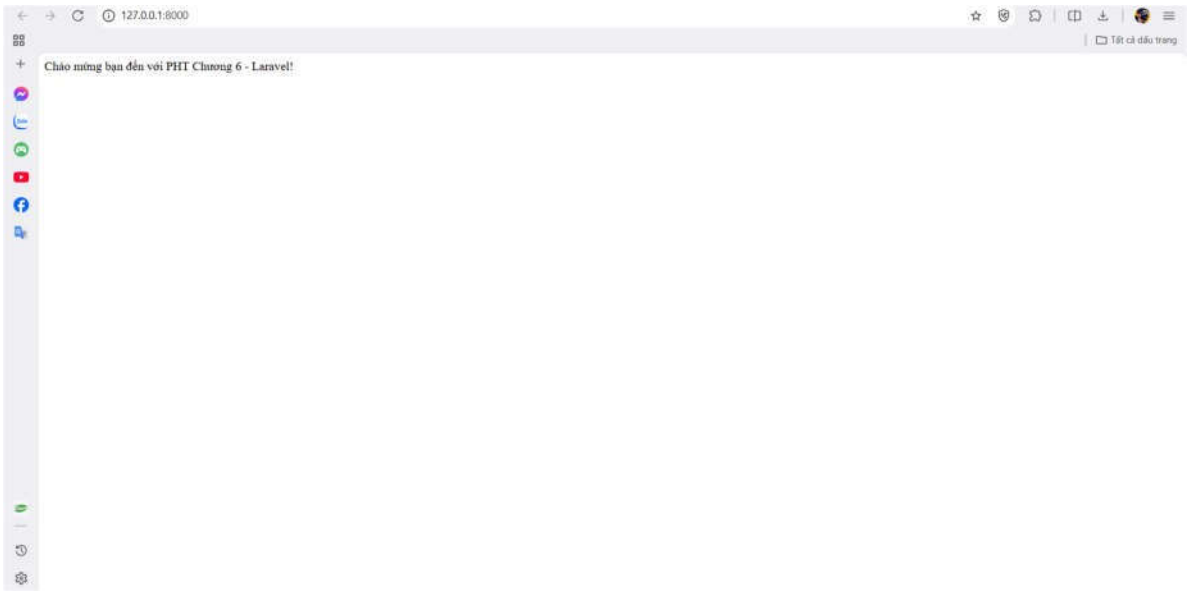
App\Http\Controllers\PageController;

Route::get('/', [PageController::class, 'showHomepage']);

Route::get('/about', [PageController::class, 'showHomepage']);








4. Câu hỏi Phản biện (Bắt buộc)

Sau khi hoàn thành Phần 2 & 3, hãy đặt 01 câu hỏi tư duy.

(Gợi ý: "Hãy so sánh file index.php (Controller) ở PHT Chương 5 và file routes/web.php ở PHT này. Tại sao Laravel lại tách 'bảng chỉ dẫn' (Routing) ra khỏi 'bộ não' (Controller)? Lợi ích của việc này là gì?").

Câu hỏi của tôi là: (Tại sao Laravel không xử lý trực tiếp logic trong file routes/web.php mà lại cần thông qua Controller? Việc tách Routing và Controller mang lại lợi ích gì cho việc bảo trì và mở rộng dự án?)

5. 골 곩 곪 곯 곬 Kết nối Đánh giá (Rất quan trọng)

Kỹ năng sử dụng Artisan, định nghĩa Route, và tạo Controller là 3 kỹ năng bắt buộc để bắt đầu làm việc với Laravel.

Đây là nền tảng cho Bài tập trên lớp (Phần Laravel) [2] (chiếm 20%, dự kiến Tuần 7) và là cấu trúc xương sống của toàn bộ Bài tập lớn theo nhóm (chiếm 50%)[3]. Nếu bạn không thể làm PHT này, bạn sẽ không thể làm được các PHT tiếp theo (Chương 7, 8).

CHƯƠNG 7: TẠO KHUÔN MẪU VỚI BLADE

1. Lý thuyết Cốt lõi (Khái niệm)

Blade là "chữ V" (View) trong mô hình MVC của Laravel. Nó là các tệp .blade.php nằm trong thư mục resources/views/.

Tại sao dùng Blade thay vì echo PHP thuần (như PHT Chương 5)?

1. Cú pháp sạch sẽ: Dùng {{ $variable }} thay vì <?php echo $variable; ?>.
2. Bảo mật (XSS): {{ ... }} tự động lọc dữ liệu (giống htmlspecialchars) để chống tấn công XSS.
3. Kế thừa Layout (Layout Inheritance): Đây là sức mạnh lớn nhất. Bạn tạo 1 "layout" (khung) chung (như menu, footer) và các trang con (như trang_chu, lien_he) chỉ cần "nhét" nội dung của mình vào khung đó.

Cú pháp Blade quan trọng:

- {{ $ten_bien }}: In ra dữ liệu (an toàn).
- {!! $ten_bien_chua_html !!}: In ra HTML (không an toàn, cẩn thận khi dùng).
- @if(...) / @elseif(...) / @else / @endif: Cấu trúc điều khiển.
- @foreach($mang as $bien) / @endforeach: Vòng lặp.
- @extends('ten_layout'): Báo cho view này biết nó "kế thừa" từ 1 file layout.
- @yield('ten_vung'): Trong file layout, đây là "lỗ hổng" (placeholder) để các view con nhét nội dung vào.
- @section('ten_vung') ... @endsection: Trong file view con, đây là khối nội dung sẽ được "nhét" vào @yield tương ứng.

2. Nhiệm vụ Thực hành (BẮT BUỘC)

Kịch bản: Chúng ta sẽ nâng cấp PHT Chương 6. Thay vì PageController trả về một chuỗi "Chào mừng...", chúng ta sẽ:

1. Tạo 1 file layout chung (app.blade.php) chứa <html>, <head>, <body>.
2. Tạo 1 file view trang chủ (homepage.blade.php) kế thừa layout đó.
3. Sửa PageController để trả về view homepage và truyền dữ liệu (như tiêu đề trang, mô tả) cho view đó.

Cấu trúc thư mục (Bắt buộc): Dự án cse485_chapter6 của bạn sẽ có 2 file view mới trong resources/views/:

/resources/views/

|-- layouts/

|  |-- app.blade.php  (Layout chính - Tệp 1)

|-- homepage.blade.php  (View trang chủ - Tệp 2)

---

Code Khởi đầu (Starter Code):

Tệp 1: resources/views/layouts/app.blade.php (Tạo file MỚI)

HTML

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{{ $title ?? 'Website Của Tôi' }}</title>
    <style>         body { font-family: sans-serif; }
        .container { max-width: 960px; margin: 0 auto; padding: 20px; }
        header, footer { background-color: #f4f4f4; padding: 10px; text-align: center; }         nav {
background-color: #333; color: white; padding: 10px; }         nav a { color: white; margin: 0 10px; }
</style>
</head>
<body>

    <header>
        <h1>Trang Web CSE485 - Chương 7</h1>
    </header>

    <nav>
        <a href="/">Trang Chủ</a>
        <a href="/about">Giới Thiệu</a>
    </nav>

    <div class="container">
        @yield('content')
    </div>

    <footer>



        <p>&copy; 2025 - Khoa CNTT - Trường Đại học Thủy Lợi</p>
    </footer>
```

```
</body>
</html>
```

Tệp 2: resources/views/homepage.blade.php (Tạo file MỚI)

PHP

```
<?php // Xóa dòng này đi, file Blade không bắt đầu bằng <?php ?>
@extends('layouts.app')



@section('content')


    <h2>{{ $page_title }}</h2>
    <p>{{ $page_description }}</p>


    <h3>Danh sách công việc (Lấy từ Controller):</h3>


    <ul>
        @foreach($tasks as $task)
            <li>{{ $task }}</li>
        @endforeach
    </ul>


@endsection
```

Tệp 3: app/Http/Controllers/PageController.php (SỬA file cũ)

PHP

```
<?php      namespace   App\Http\Controllers;
use Illuminate\Http\Request;
```

```
class PageController extends Controller
{
   // Sửa phương thức showHomepage() của Chương 6     public function
showHomepage()
   {
       // TODO 7: Thay vì "return 'Chào mừng...'", chúng ta sẽ
// chuẩn bị dữ liệu để truyền cho View


       $viewTitle = 'PHT Chương 7 - Blade Template';
       $pageTitle = 'Chào mừng bạn đến với Blade!';
       $pageDescription = 'Đây là trang chủ được render bằng Blade Template
Engine.';
       $tasks = [
           'Cài đặt Laravel',
           'Hiểu về Routing & Controller',
           'Tạo Layout với Blade',
           'Truyền dữ liệu cho View'
       ];


       // TODO 8: Trả về một View
       // Gợi ý: Dùng hàm view('ten_view',
$data_array)          // 'homepage' tương đương
'homepage.blade.php'                  return
view('homepage', [              'title' => $viewTitle,
           'page_title' => $pageTitle,
           'page_description' => $pageDescription,
           'tasks' => $tasks
       ]);


       // TODO 9: (Cách khác) Dùng hàm compact() cho gọn
       // return view('homepage', compact('viewTitle', 'pageTitle',
'pageDescription', 'tasks'));
       // (Lưu ý: khi dùng compact, tên biến ở Controller và View phải khớp nhau)


   }
```

```
}
```

3. Yêu cầu Bằng chứng (Proof of Work) Bạn phải

nộp lại 2 bằng chứng sau:

A. Code đã hoàn thiện: Dán (paste) toàn bộ code của 3 tệp bạn đã tạo/sửa:

1. resources/views/layouts/app.blade.php
2. resources/views/homepage.blade.php
3. app/Http/Controllers/PageController.php

B. Ảnh chụp màn hình Kết quả (Trình duyệt Web): Chạy php artisan serve và truy cập http://127.0.0.1:8000/. Chụp ảnh màn hình trình duyệt hiển thị kết quả. (Phải thấy rõ layout chung (header/footer) và phần nội dung (danh sách công việc) được nạp ở giữa). (Dán Code A1, A2, A3 và Ảnh B của bạn vào đây)

A1

```
<!DOCTYPE html>
<html lang="vi">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{{ $title ?? 'Website Của Tôi' }}</title>
    <style>
        body { font-family: sans-serif; }
        .container { max-width: 960px; margin: 0 auto; padding: 20px; }
        header, footer { background-color: #f4f4f4; padding: 10px; text-align: center; }
nav { background-color: #333; color: white; padding: 10px; }         nav a { color:
white; margin: 0 10px; }
    </style>
</head>
<body>

    <header>
        <h1>Trang Web CSE485 - Chương 7</h1>
    </header>

   <nav>
```

```
        <a href="/">Trang Chủ</a>
        <a href="/about">Giới Thiệu</a>
    </nav>

    <div class="container">
        @yield('content')
    </div>

    <footer>
        <p>&copy; 2025 - Khoa CNTT - Trường Đại học Thủy Lợi</p>
    </footer>

</body>
</html>
```

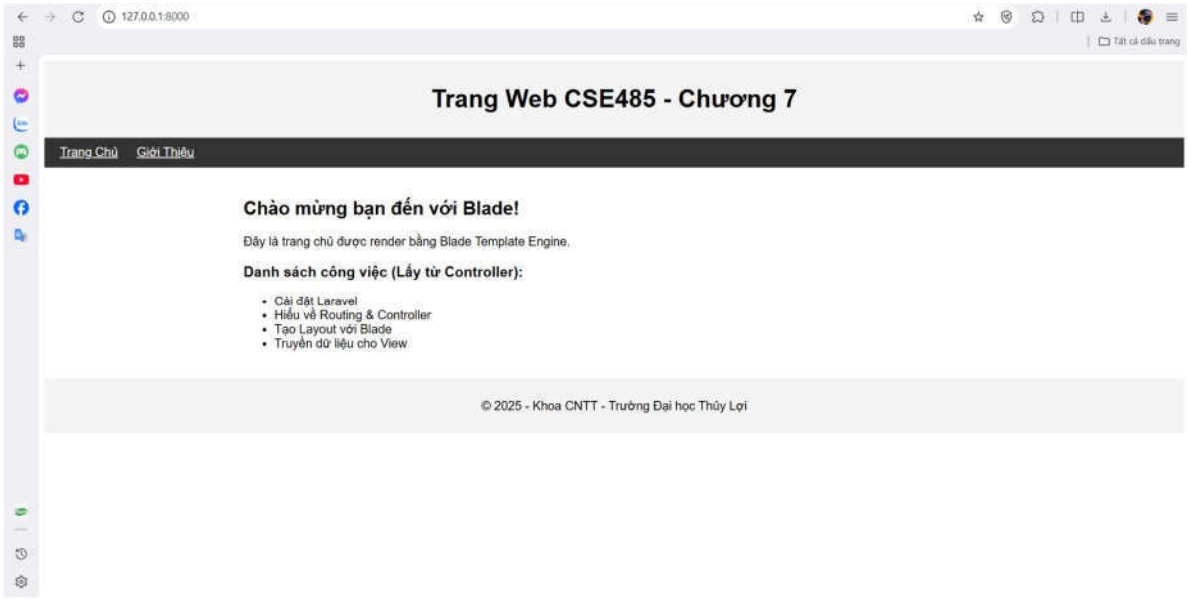
A2

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

@section('content')

    <h2>{{ $page_title }}</h2>
    <p>{{ $page_description }}</p>

    <h3>Danh sách công việc (Lấy từ Controller):</h3>

    <ul>
        @foreach($tasks as $task)
            <li>{{ $task }}</li>
        @endforeach
    </ul>
```

@endsection

A3

```
<?php

namespace App\Http\Controllers;

use Illuminate\Http\Request;

class PageController extends Controller
{
   public function showHomepage()
   {
      $viewTitle = 'PHT Chương 7 - Blade Template';
      $pageTitle = 'Chào mừng bạn đến với Blade!';
      $pageDescription = 'Đây là trang chủ được render bằng Blade Template Engine.';
$tasks = [
          'Cài đặt Laravel',
          'Hiểu về Routing & Controller',
          'Tạo Layout với Blade',
          'Truyền dữ liệu cho View'
       ];

      return view('homepage', [
          'title' => $viewTitle,
          'page_title' => $pageTitle,
          'page_description' => $pageDescription,
         'tasks' => $tasks
      ]);
   }
}
```




4. Câu hỏi Phản biện (Bắt buộc)

Sau khi hoàn thành Phần 2 & 3, hãy đặt 01 câu hỏi tư duy.

(Gợi ý: "Hãy giải thích chính xác vai trò của @yield('content') (trong file app.blade.php) và @section('content') (trong file homepage.blade.php). Chúng làm việc cùng nhau như thế nào để ghép thành file HTML cuối cùng mà trình duyệt nhận được?").

Câu hỏi của tôi là: (@yield('content') và @section('content') phối hợp với nhau như thế nào để Laravel ghép layout và nội dung thành một trang HTML hoàn chỉnh gửi về cho trình duyệt?)

5. 곬곮곩골곫 Kết nối Đánh giá (Rất quan trọng)

Kỹ năng sử dụng Blade (tách layout, truyền dữ liệu) là kỹ năng bắt buộc để xây dựng giao diện người dùng (Frontend) trong Laravel.

Bạn sẽ vận dụng trực tiếp PHT này để hoàn thành Bài tập trên lớp (Phần Laravel), chiếm 20% tổng điểm, dự kiến vào Tuần 7. Nó cũng là phần "V" (View) không thể thiếu trong Bài tập lớn theo nhóm (50%).

## CHƯƠNG 8: ELOQUENT ORM

1. Lý thuyết Cốt lõi (Khái niệm)

- ORM là gì? (Object-Relational Mapping). Đây là một kỹ thuật "ánh xạ" (mapping) một bảng (table) trong CSDL (ví dụ: bảng users) thành một "Đối tượng" (Object) trong code (ví dụ: class User của PHP).
- Eloquent (8.1): Là ORM của Laravel.
    - Model: Mỗi bảng trong CSDL của bạn sẽ có một file Model tương ứng trong thư mục app/Models/. Ví dụ: Bảng sinhviens $\leftrightarrow$ Model app/Models/SinhVien.php.

- Quy ước (Convention): Eloquent rất thông minh. Nếu bạn đặt tên Model là SinhVien (số ít, viết hoa chữ cái đầu), nó sẽ tự động hiểu rằng Model này làm việc với bảng sinhviens (số nhiều, viết thường).

- Migrations (8.2): Là "hệ thống quản lý phiên bản" cho CSDL của bạn. Thay vì tạo bảng bằng tay trong phpMyAdmin, bạn sẽ viết code PHP (trong thư mục database/migrations/) để định nghĩa cấu trúc bảng. o Lệnh tạo: php artisan make:model TenModel -m (cờ -m sẽ tạo luôn file migration). o Lệnh chạy: php artisan migrate (để tạo bảng trong CSDL).
- Truy vấn (8.3, 8.4): Đây là điều tuyệt vời nhất.
  - Thay vì PDO (Chương 4):

```
$sql = "SELECT * FROM sinhviens";
```

  - Dùng Eloquent (Chương 8):

```
$all_students = SinhVien::all();
```

  - Thay vì INSERT (Chương 4):

```
$sql = "INSERT INTO sinhviens (ten, email) VALUES (?, ?)";
```

  - Dùng Eloquent (Chương 8):

```
SinhVien::create(['ten_sinh_vien' => $ten, 'email' => $email]);
```

2. Nhiệm vụ Thực hành (BẮT BUỘC)

Kịch bản: Chúng ta sẽ "nâng cấp" PHT Quản lý sinh viên (từ Chương 4 & 5) lên chuẩn Laravel 100%, sử dụng Eloquent Model và Migrations.

---

A. Thiết lập Database & Migration (8.2)

1. // TODO 1: Mở file .env ở thư mục gốc dự án (cse485_chapter6).
2. // TODO 2: Sửa các thông số CSDL. Tạo một CSDL mới trong phpMyAdmin tên là cse485_laravel (hoặc tên khác tùy bạn).

Đoạn mã

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=cse485_laravel  # Tên CSDL bạn vừa tạo
DB_USERNAME=root            # User XAMPP
DB_PASSWORD=                # Pass XAMPP (rỗng)
```

3. // TODO 3: Mở Terminal, cd vào dự án, chạy lệnh Artisan sau để tạo Model SinhVien VÀ file migration (-m): Bash php artisan make:model SinhVien -m
4. // TODO 4: Mở file migration vừa được tạo trong:

database/migrations/..._create_sinh_viens_table.php (tên file có thêm timestamp).

5. // TODO 5: Sửa hàm up() để định nghĩa cấu trúc bảng (thêm 2 cột ten_sinh_vien và email): PHP

```
public function up(): void
{
   Schema::create('sinh_viens', function (Blueprint $table) {
      $table->id(); // Tự động tạo cột 'id' (bigint, auto-increment, primary key)

      // TODO 5: Thêm 2 dòng này
      $table->string('ten_sinh_vien', 255);
      $table->string('email', 255)->unique(); // unique() là ràng buộc duy nhất

      $table->timestamps(); // Tự động tạo 2 cột 'created_at' và 'updated_at'
   });  }
```

6. // TODO 6: Chạy lệnh migrate để tạo bảng trong CSDL:

Bash php artisan migrate

(Vào phpMyAdmin kiểm tra CSDL cse485_laravel, bạn sẽ thấy bảng sinh_viens xuất hiện).

---

B. Cập nhật Model & Controller (8.3, 8.4)

1. // TODO 7: Mở file Model app/Models/SinhVien.php.
2. // TODO 8: Để cho phép ::create() (Mass Assignment), chúng ta phải khai báo use HasFactory; và các trường được phép điền vào mảng $fillable:

PHP

```
<?php namespace App\Models; use
Illuminate\Database\Eloquent\Factories\HasFactory; use
Illuminate\Database\Eloquent\Model;   class SinhVien extends Model
{    use HasFactory;

   // TODO 8: Thêm mảng $fillable     protected $fillable =
[
      'ten_sinh_vien',
      'email',
   ];
}
```

3. // TODO 9: (Tùy chọn) Tạo 1 Controller mới cho Sinh Viên: Bash php artisan

   make:controller SinhVienController

4. // TODO 10: Mở app/Http/Controllers/SinhVienController.php và use Model SinhVien ở đầu file. Sau đó tạo 2 phương thức index() (để hiển thị) và store() (để lưu):

PHP

```
<?php namespace App\Http\Controllers; use Illuminate\Http\Request;


// TODO 10: Import Model SinhVien use

App\Models\SinhVien;
```

```
class SinhVienController extends Controller
{
   // Phương thức index() (SELECT)     public function
index()
   {
      // TODO 11: Dùng Eloquent ::all() để lấy toàn bộ sinh viên
// Gợi ý: $danhSachSV = SinhVien::all();


      // TODO 12: Trả về 1 view 'sinhvien.list' và truyền $danhSachSV
      // Gợi ý: return view('...', compact('...'));
   }


   // Phương thức store() (INSERT)     public function
store(Request $request)
   {
      // TODO 13: Lấy toàn bộ dữ liệu từ form
      // Gợi ý: $data = $request->all();


      // TODO 14: Dùng Eloquent ::create() để lưu vào CSDL
      // (Lưu ý: tên input trong form phải khớp với $fillable và tên cột)
// Gợi ý: SinhVien::create($data);


      // TODO 15: Chuyển hướng về trang danh sách
      // Gợi ý: return redirect()->route('sinhvien.index');
   }
}
```

5. // TODO 16: Mở routes/web.php, import SinhVienController và tạo 2 route (1 GET, 1 POST): PHP

```
// Gợi ý:
// use App\Http\Controllers\SinhVienController;
// Route::get('/sinhvien', [SinhVienController::class, 'index'])-
>name('sinhvien.index');
// Route::post('/sinhvien', [SinhVienController::class, 'store'])-
>name('sinhvien.store');
```

6. // TODO 17: Tạo file View resources/views/sinhvien/list.blade.php (tạo thư mục sinhvien nếu chưa có). Copy/paste code từ PHT Chương 7 (layout, form, table) và sửa lại: o Form: action="{{ route('sinhvien.store') }}" và method="POST". Thêm @csrf (Bắt buộc cho form POST của Laravel - sẽ học ở Chương 9). Input name phải là ten_sinh_vien và email.

o Table: Dùng @foreach($danhSachSV as $sv) và {{ $sv->ten_sinh_vien }}. 3. Yêu cầu

Bằng chứng (Proof of Work) Bạn phải nộp lại 3 bằng chứng sau: A. Code đã hoàn

thiện:

1. Dán (paste) code hàm up() trong file Migration (..._create_sinh_viens_table.php).
2. Dán (paste) toàn bộ code file app/Models/SinhVien.php.
3. Dán (paste) toàn bộ code file app/Http/Controllers/SinhVienController.php.
4. Dán (paste) code 2 route trong routes/web.php. B. Ảnh chụp màn hình Kết quả (3 ẢNH):

1. Ảnh 1 (Terminal): Chụp màn hình Terminal sau khi chạy php artisan make:model ... -m VÀ php artisan migrate thành công.
2. Ảnh 2 (phpMyAdmin): Chụp màn hình tab "Structure" (Cấu trúc) của bảng sinh_viens trong CSDL, cho thấy rõ các cột id, ten_sinh_vien, email, created_at.
3. Ảnh 3 (Trình duyệt Web): Chụp ảnh màn hình trang /sinhvien, sau khi đã dùng form thêm 1-2 sinh viên (chứng minh ::create() và ::all() đều hoạt động). (Dán Code A1, A2, A3, A4 và 3 Ảnh B1, B2, B3 của bạn vào đây)
4. Câu hỏi Phản biện (Bắt buộc)

Sau khi hoàn thành Phần 2 & 3, hãy đặt 01 câu hỏi tư duy.

1.

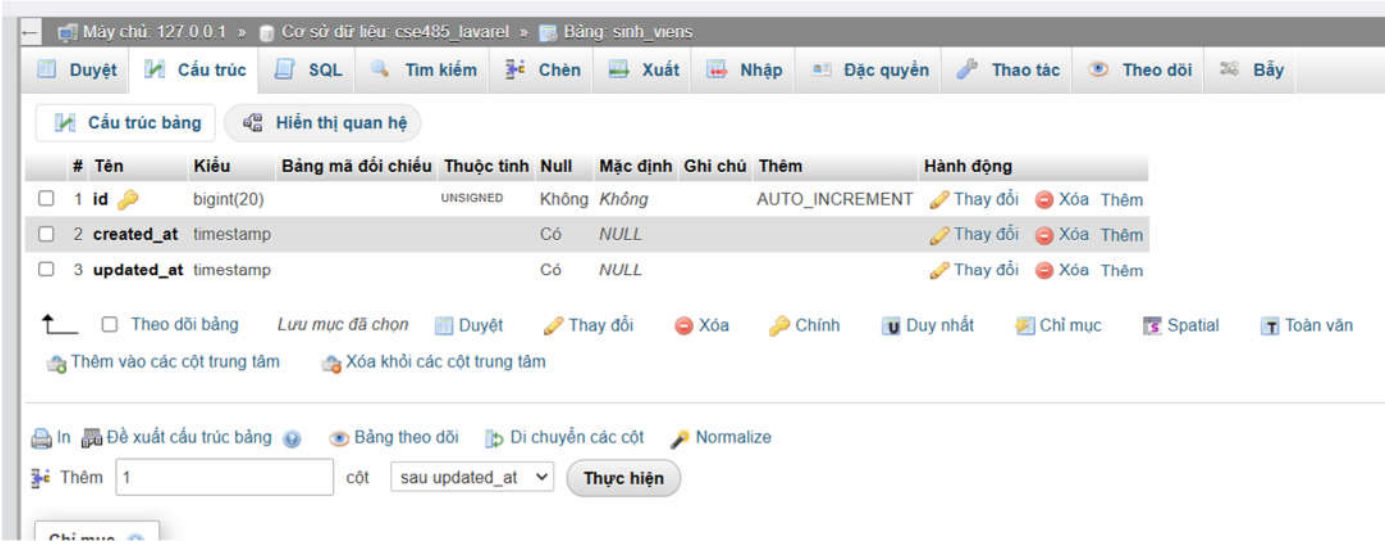
```
PS C:\xampp\htdocs\cse485_chapter6> php artisan make:model SinhVien

  INFO  Model [C:\xampp\htdocs\cse485_chapter6\app\Models\SinhVien.php] created successfully.

PS C:\xampp\htdocs\cse485_chapter6> php artisan make:model SinhVien -m

  INFO  Model [C:\xampp\htdocs\cse485_chapter6\app\Models\SinhVien.php] created successfully.

  INFO  Migration [C:\xampp\htdocs\cse485_chapter6\database\migrations\2025_12_29_034008_create_sinh_viens_table.php] created successfully.

PS C:\xampp\htdocs\cse485_chapter6>
```

1.

```
public function up(): void
{
    Schema::create('sinh_viens', function (Blueprint $table) {
        $table->id();
        $table->string('ten_sinh_vien', 255);
        $table->string('email', 255)->unique();
```

```
        $table->timestamps();
    });
}
```

2.




2. <?php

```
namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Factories\HasFactory;

class SinhVien extends Model
{
    use HasFactory;

   protected $fillable = [
       'ten_sinh_vien',
       'email',
   ];
}
```

3. <?php

```
namespace App\Http\Controllers;

use Illuminate\Http\Request;
use App\Models\SinhVien;

class SinhVienController extends Controller
{
    public function index()
    {
        $danhSachSV = SinhVien::all();
        return view('sinhvien.list', compact('danhSachSV'));
    }

    public function store(Request $request)
    {
        SinhVien::create($request->all());
        return redirect()->route('sinhvien.index');
    }
}
```

4.
```
<?php

use App\Http\Controllers\SinhVienController;
use Illuminate\Support\Facades\Route;

Route::get('/sinhvien', [SinhVienController::class, 'index'])
    ->name('sinhvien.index');

Route::post('/sinhvien', [SinhVienController::class, 'store'])
    ->name('sinhvien.store');
```

3.


(Gợi ý: "Hãy so sánh PHT này (Eloquent) với PHT Chương 4 (PDO). Lợi ích lớn nhất của Eloquent là gì? Khi nào (trường hợp nào) thì dùng Eloquent sẽ tiện lợi hơn, và khi nào thì dùng PDO/SQL thuần có thể sẽ tốt hơn?").

Câu hỏi của tôi là: (So với việc sử dụng PDO thuần ở Chương 4, Eloquent ORM giúp giảm đáng kể lượng code SQL và tăng khả năng bảo trì ứng dụng. Vậy trong các hệ thống lớn, khi nào nên ưu tiên dùng Eloquent và khi nào nên kết hợp hoặc chuyển sang dùng SQL thuần để tối ưu hiệu năng?)

5. 골 굵 곪 곯 궀 Kết nối Đánh giá (Rất quan trọng)

Eloquent ORM (Chương 8) là "trái tim" xử lý dữ liệu của Laravel. Kỹ năng make:model, migrate, và sử dụng các phương thức ::all(), ::find(), ::create() là tối quan trọng.

Đây là kiến thức trọng tâm của cả Bài tập trên lớp (Phần Laravel) (20%) [2]và Bài tập lớn theo nhóm (50%)[3]. Nếu không nắm vững Eloquent, bạn không thể xây dựng bất kỳ chức năng nào cho dự án của mình.

## CHƯƠNG 9: BẢO MẬT ỨNG DỤNG WEB [1]

1. Lý thuyết Cốt lõi (Khái niệm)

Trong chương này, chúng ta tập trung vào 4 khái niệm bảo mật chính:

- XSS (Cross-Site Scripting) (9.4):
    - Vấn đề: Kẻ tấn công tìm cách "tiêm" (inject) một đoạn mã JavaScript độc hại (ví dụ: qua ô comment, ô nhập tên) vào CSDL của bạn. Khi một người dùng khác tải trang đó, mã JS độc hại sẽ chạy trên trình duyệt của họ và đánh cắp thông tin (như cookie đăng nhập).
    - Giải pháp Laravel: Cỗ máy Blade tự động bảo vệ bạn. Khi bạn dùng cú pháp {{ $variable }}, Laravel sẽ "thoát" (escape) tất cả các thẻ HTML, biến

<script>alert(1)</script> thành text vô hại <script>alert(1)</script; (sẽ được trình duyệt in ra y hệt, chứ không chạy).

- CSRF (Cross-Site Request Forgery) (9.4):
  - Vấn đề: Kẻ tấn công lừa trình duyệt của một người dùng (đang đăng nhập vào trang của bạn) gửi một yêu cầu đến trang của bạn mà người dùng không hề hay biết. Ví dụ: Kẻ tấn công đặt một ảnh "vô hình" trên trang web của hắn: <img src="http://yoursite.com/delete-user?id=123" width="1" height="1">. Khi người dùng (đã đăng nhập) truy cập trang của kẻ tấn công, trình duyệt sẽ tự động tải ảnh này, vô tình gửi yêu cầu xóa tài khoản.
  - Giải pháp Laravel: Laravel yêu cầu mọi form POST, PUT, PATCH, DELETE phải đính kèm một "token" (mã bí mật) duy nhất cho phiên đó.

    Bạn chỉ cần thêm directive @csrf vào trong form.

    Laravel sẽ tự động tạo thẻ <input type="hidden" name="_token" value="...token...">.

    Khi form được gửi, Laravel kiểm tra token này. Nếu token không khớp (hoặc không có), nó sẽ từ chối yêu cầu (lỗi 419 Page Expired).
- Xác thực (Authentication) (9.3):
  - Trả lời câu hỏi: "Bạn là ai?"
  - Đây là quá trình định danh người dùng, ví dụ: kiểm tra username/password (chức năng Đăng nhập, Đăng ký).
- Phân quyền (Authorization) (9.3):
  - Trả lời câu hỏi: "Bạn được phép làm gì?"
  - Xảy ra sau khi đã xác thực. Ví dụ: Người dùng A (đã đăng nhập) là "user" nên chỉ được xem bài viết, nhưng người dùng B (đã đăng nhập) là "admin" thì được quyền sửa và xóa bài viết.

2. Nhiệm vụ Thực hành (BẮT BUỘC)

Kịch bản: Chúng ta sẽ nâng cấp PHT Chương 8 (Quản lý sinh viên) để vá 2 lỗ hổng bảo mật lớn là CSRF và XSS.

Code Khởi đầu (Starter Code): Sử dụng dự án và các tệp của PHT Chương 8.

---

A. Chống tấn công CSRF (9.4)

1. // TODO 1: Mở tệp View resources/views/sinhvien/list.blade.php (từ PHT 8).
2. // TODO 2: Tìm thẻ <form ...> (dùng để thêm sinh viên).
3. // TODO 3: Thêm directive @csrf ngay bên dưới thẻ mở <form ...>: HTML

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


  Tên sinh viên: <input type="text" name="ten_sinh_vien" required>

  Email: <input type="email" name="email" required>
```

```
  <button type="submit">Thêm</button>
</form>
```

---

B. Kiểm chứng khả năng chống XSS (9.4)

1. // TODO 4: Đảm bảo rằng trong vòng lặp @foreach của tệp list.blade.php, bạn đang dùng cú pháp {{ }} (hai dấu ngoặc nhọn) để in tên sinh viên, không phải {!! !!} (một nhọn, hai chấm than). PHP

```
@foreach($danhSachSV as $sv)
  <tr>
    <td>{{ $sv->id }}</td>
    <td>{{ $sv->ten_sinh_vien }}</td>
    <td>{{ $sv->email }}</td>
  </tr>
@endforeach
```

2. // TODO 5: Chạy php artisan serve và mở trang /sinhvien.
3. // TODO 6: Sử dụng form, thêm một sinh viên mới với thông tin sau:
   - Tên sinh viên: <script>alert('Ban da bi XSS!');</script>
   - Email: hacker@email.com
4. // TODO 7: Nhấn "Thêm" và quan sát bảng danh sách sinh viên.

---

C. Bảo vệ Route bằng Xác thực (9.3) (Giới thiệu)

1. // TODO 8: Mở file routes/web.php.
2. // TODO 9: Giả sử bạn đã cài đặt hệ thống đăng nhập (ví dụ: Laravel Breeze), bạn có thể "bọc" các route sinh viên bằng một middleware để bắt buộc người dùng phải đăng nhập mới được xem. PHP

```
// Gợi ý:
// Route::middleware(['auth'])->group(function () {
//     Route::get('/sinhvien', [SinhVienController::class, 'index'])->name('sinhvien.index');
//     Route::post('/sinhvien', [SinhVienController::class, 'store'])->name('sinhvien.store');
// });
// 'auth' là middleware kiểm tra xác thực
```

(Đây là phần giới thiệu, bạn sẽ phải làm điều này trong BTL)

3. Yêu cầu Bằng chứng (Proof of Work) Bạn phải nộp lại 2 bằng chứng sau: A. Code đã hoàn thiện:

1. Dán (paste) code của khối <form> trong tệp list.blade.php (chứng minh bạn đã thêm @csrf).

```
<form method="POST" action="{{ route('sinhvien.store') }}">
    @csrf
    <input name="ten_sinh_vien" placeholder="Tên sinh viên">
    <input name="email" placeholder="Email">
    <button type="submit">Thêm</button>
</form>
```

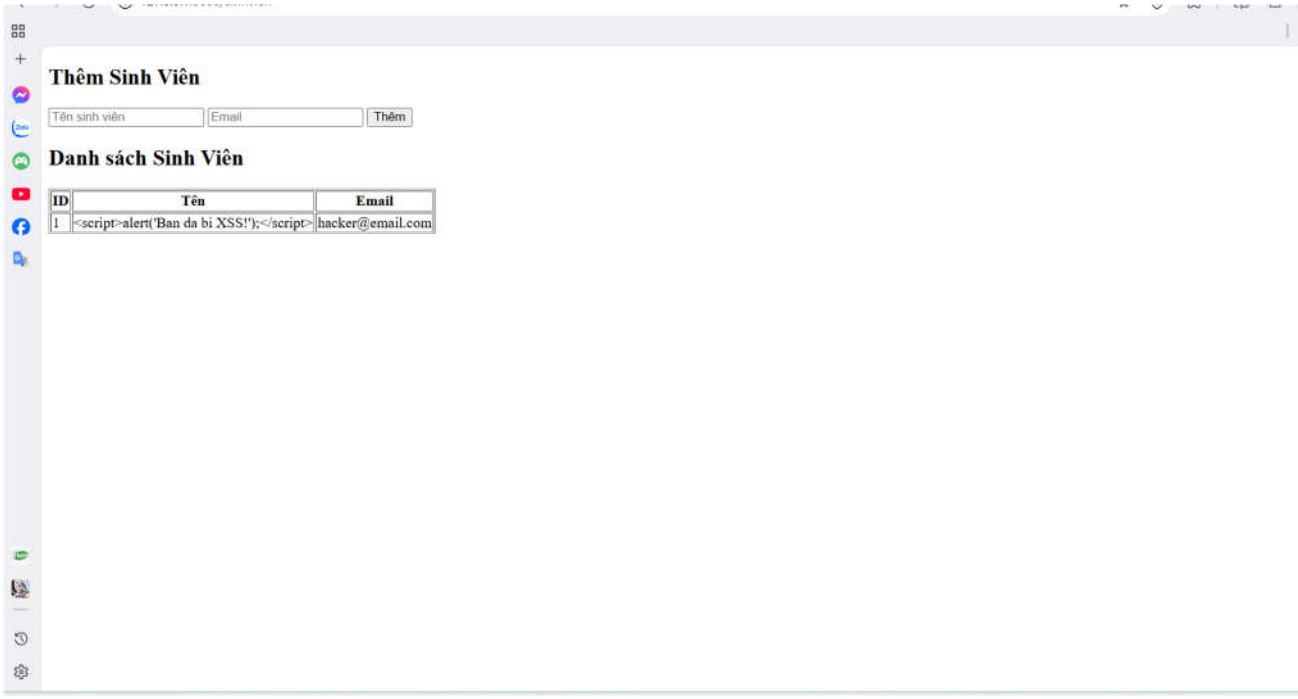
1. Dán (paste) code của khối @foreach trong tệp list.blade.php (chứng minh bạn dùng {{ }}).

```
@foreach($danhSachSV as $sv)
    <tr>
        <td>{{ $sv->id }}</td>
        <td>{{ $sv->ten_sinh_vien }}</td>
        <td>{{ $sv->email }}</td>
    </tr>
    @endforeach
```

B. Ảnh chụp màn hình Kết quả (BẮT BUỘC 2 ẢNH):

1. Ảnh 1 (Bằng chứng Chống CSRF): Tải trang /sinhvien, nhấn chuột phải $\rightarrow$ View Page Source (Xem nguồn trang). Chụp ảnh màn hình mã nguồn HTML, khoanh tròn vào thẻ <input type="hidden" name="_token" ...> mà @csrf đã tự động tạo ra.


2. Ảnh 2 (Bằng chứng Chống XSS): Chụp ảnh màn hình trang /sinhvien sau khi bạn đã thêm sinh viên ở (TODO 6 & 7). Ảnh phải cho thấy dòng chữ <script>alert('Ban da bi XSS!');</script> được in ra dưới dạng text trên bảng, chứ KHÔNG CÓ popup "alert" nào hiện lên.




(Dán Code A1, A2 và 2 Ảnh B1, B2 của bạn vào đây)

4. Câu hỏi Phản biện (Bắt buộc)

Sau khi hoàn thành PHT, hãy đặt 01 câu hỏi tư duy.

(Gợi ý: "Sự khác biệt cơ bản giữa Xác thực (Authentication) và Phân quyền (Authorization) là gì? [2] Trong Bài tập lớn, chức năng 'Đăng nhập' là Authentication hay Authorization? Chức năng 'Chỉ Admin mới thấy trang Quản trị' là gì?").

Câu hỏi của tôi là: (Sự khác biệt giữa Authentication và Authorization là gì?
Trong Bài tập lớn, chức năng Đăng nhập thuộc Authentication hay Authorization?
Và chức năng chỉ cho phép Admin truy cập trang Quản trị thuộc loại nào?)

5. 골 괽 굟 괇 괈 Kết nối Đánh giá (Rất quan trọng) Kiến thức trong PHT này là tối

quan trọng. 괈 골 괽 굟 괇 Áp dụng: Bảo mật (chống XSS, CSRF) và Xác thực/Phân

quyền

(Authentication/Authorization) là các yêu cầu BẮT BUỘC và chiếm trọng số điểm cao trong Bài tập lớn theo nhóm (50%)[3]. Một dự án nộp cho giảng viên mà không có @csrf trong form, bị lỗi XSS, hoặc không bảo vệ các trang quản trị (cho phép truy cập mà không cần đăng nhập) sẽ bị xem là lỗi nghiêm trọng và bị trừ điểm rất nặng.