ADMIN PANEL CINEBOOK - SCRIPT THUYẾT TRÌNH CHI TIẾT

PHẦN 4: CÂU HỎI VẤN ĐÁP HỘI ĐỒNG VÀ CÁCH TRẢ LỜI

MỤC LỤC

1. CÂU HỎI VỀ KIẾN TRÚC & DESIGN PATTERNS

Q1.1: Tại sao bạn chọn MVC pattern cho dự án này?

Cách trả lời ăn điểm:

> "Em chọn MVC pattern vì ba lý do chính:
>
> **Thứ nhất**, MVC giúp tách biệt rõ ràng 3 tầng: Model xử lý dữ liệu, View hiển thị giao diện, Controller điều phối logic. Điều này giúp code dễ bảo trì và mở rộng.
>
> **Thứ hai**, Laravel - framework em sử dụng - được thiết kế theo MVC, nên việc áp dụng pattern này rất tự nhiên và tận dụng được tối đa sức mạnh của framework.
>
> **Thứ ba**, với dự án đặt vé xem phim có nhiều entity (User, Movie, Booking...), MVC giúp tổ chức code theo từng module rõ ràng. Ví dụ: AdminMovieController xử lý logic phim, AdminBookingController xử lý logic đặt vé..."

Q1.2: Giải thích cấu trúc thư mục Admin của bạn?

Cách trả lời:

> "Cấu trúc Admin em tổ chức như sau:
>
> ```
> Controllers/Admin/     - Tất cả controller admin tách riêng
> views/admin/           - Views riêng cho admin
> Middleware/CheckRole   - Middleware phân quyền
> ```

Em tách Admin thành namespace riêng vì:

1. **Separation of Concerns**: Admin và User có logic khác nhau
2. **Security**: Dễ apply middleware cho toàn bộ admin routes
3. **Maintainability**: Team có thể làm việc song song trên admin và frontend"

Q1.3: Tại sao tách riêng các Controller cho Admin thay vì dùng chung với frontend?

Cách trả lời:

"Có 3 lý do chính:

1. Khác biệt về chức năng:

- Frontend: User chỉ xem phim, đặt vé
- Admin: Cần full CRUD, thống kê, quản lý

2. Khác biệt về authorization:

- Frontend routes: Public hoặc auth
- Admin routes: auth + role:admin

3. Khác biệt về response:

- Frontend: Cần UI đẹp, user-friendly
- Admin: Cần data-rich, tables, forms

Nếu gộp chung sẽ có nhiều if-else kiểm tra role, code khó đọc và dễ có lỗi bảo mật."

---

2. CÂU HỎI VỀ DATABASE & ELOQUENT

Q2.1: Giải thích về quan hệ Many-to-Many giữa Movie và Genre?

Cách trả lời:

"Movie và Genre có quan hệ many-to-many:

- Một phim có thể thuộc nhiều thể loại (Action, Sci-fi...)
- Một thể loại có nhiều phim

Em implement bằng **pivot table** `movie_genre`:

```
CREATE TABLE movie_genre (
    movie_id    -- FK tới movies
    genre_id    -- FK tới genres
);
```

Trong Eloquent:

```php
// Movie.php
public function genres() {
    return $this->belongsToMany(Genre::class, 'movie_genre');
}
```

Khi tạo/update movie, em dùng `sync()`:

```php
$movie->genres()->sync([1, 2, 3]);  // Cập nhật thể loại
```

`sync()` thông minh hơn `attach()` vì nó tự xóa các genre cũ không còn trong list."

Q2.2: N+1 Query Problem là gì? Bạn xử lý như thế nào?

Cách trả lời:

"**N+1 Problem** xảy ra khi query trong loop:

```php
// ❌ N+1 PROBLEM
$bookings = Booking::all();  // 1 query
foreach ($bookings as $booking) {
    echo $booking->user->name;  // N queries
}
// Tổng: 1 + N queries (N = số booking)
```

Giải pháp: Eager Loading

```php
// ☑ GIẢI PHÁP
$bookings = Booking::with(['user', 'showtime.movie'])->get();
// Chỉ có 3-4 queries dù có 1000 booking
```

Trong dự án em, ở `AdminBookingController@index`:

```php
$query = Booking::with([
    'user',
    'showtime.movie',
    'showtime.room',
    'bookingSeats.seat'
]);
```

Em cũng dùng `withCount()` cho thống kê:

```
$users = User::withCount('bookings')->get();
// $user->bookings_count available mà không cần load tất cả bookings
```"

Q2.3: Tại sao cần Database Transaction? Ví dụ cụ thể?

Cách trả lời:

> "Transaction đảm bảo **ACID** - đặc biệt là **Atomicity**: tất cả thao tác phải thành công, hoặc không có gì xảy ra.
>
> **Ví dụ cụ thể trong dự án**: Khi tạo Showtime mới:

```php
DB::beginTransaction();
try {
    // Bước 1: Tạo showtime
    $showtime = Showtime::create([...]);

    // Bước 2: Tạo giá cho 3 loại ghế
    ShowtimePrice::create([...]);  // Standard
    ShowtimePrice::create([...]);  // VIP
    ShowtimePrice::create([...]);  // Couple

    // Bước 3: Tạo 100 showtime_seats
    foreach ($room->seats as $seat) {
        ShowtimeSeat::create([...]);
    }

    DB::commit();  // Thành công
} catch (Exception $e) {
    DB::rollBack();  // Lỗi → hoàn tác tất cả
}
```

> Nếu không có transaction, khi lỗi ở bước 3 (sau khi tạo 50 seats), database sẽ có:
>
> - 1 showtime ✓
> - 3 prices ✓
> - 50/100 seats → **Data không nhất quán!**
>
> Với transaction, nếu lỗi → tất cả bị rollback → database sạch."

Q2.4: Giải thích query tính doanh thu theo phim trong Dashboard?

Cách trả lời:

> "Query này tính tổng doanh thu mỗi phim:

```
$movieRevenue = Movie::leftJoin('showtimes', 'movies.id', '=',
'showtimes.movie_id')
    ->leftJoin('bookings', function ($join) {
        $join->on('showtimes.id', '=', 'bookings.showtime_id')
            ->where('bookings.payment_status', 'paid');
    })
    ->select('movies.id', 'movies.title',
             DB::raw('COALESCE(SUM(bookings.total_price), 0) as revenue'))
    ->groupBy('movies.id', 'movies.title')
    ->orderBy('revenue', 'desc')
    ->get();
```

Giải thích:

1. **LEFT JOIN**: Để lấy cả phim chưa có booking (revenue = 0)
2. **Closure trong join**: Chỉ join bookings đã thanh toán
3. **COALESCE**: Biến NULL thành 0 cho phim chưa có doanh thu
4. **GROUP BY**: Nhóm theo phim để SUM

Kết quả: Danh sách phim sắp xếp theo doanh thu cao → thấp."

---

3. CÂU HỎI VỀ SECURITY

Q3.1: Hệ thống phân quyền Admin hoạt động như thế nào?

Cách trả lời:

"Hệ thống phân quyền gồm 2 lớp:

Lớp 1: Authentication (Xác thực)

- Middleware `auth` kiểm tra user đã đăng nhập chưa
- Nếu chưa → redirect về login

Lớp 2: Authorization (Phân quyền)

- Middleware `role:admin` kiểm tra role
- Code trong `CheckRole.php`:

```
public function handle($request, Closure $next, string $role)
{
    if (!Auth::check()) {
        return redirect()->route('login');
    }

    if (Auth::user()->role !== $role) {
        abort(403, 'Unauthorized');
    }
```

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

Apply trong routes/web.php:

```
Route::prefix('admin')
    ->middleware(['auth', 'role:admin'])  // 2 middleware
    ->group(function () {
        // Tất cả routes admin
    });
```

Nếu user thường vào `/admin/*` → 403 Forbidden."

Q3.2: Làm sao tránh SQL Injection trong dự án?

Cách trả lời:

"Em phòng chống SQL Injection bằng 3 cách:

1. Sử dụng Eloquent ORM:

```
// ☑ AN TOÀN - Eloquent tự escape
User::where('email', $request->email)->first();
```

2. Query Builder với binding:

```
// ☑ AN TOÀN - Parameterized query
DB::table('users')->where('name', 'like', '%' . $search . '%');
```

3. Tuyệt đối không dùng raw query với input:

```
// ❌ NGUY HIỂM - Không bao giờ làm
DB::select(\"SELECT * FROM users WHERE email = '$email'\");
```

Eloquent và Query Builder trong Laravel sử dụng **PDO prepared statements**, tự động escape input nên rất an toàn."

Q3.3: Giải thích CSRF Protection trong forms?

Cách trả lời:

"**CSRF (Cross-Site Request Forgery)** là tấn công giả mạo request từ user đã đăng nhập.

Ví dụ tấn công:

```
<!-- Trang web độc hại -->
<img src=\"https://cinebook.com/admin/users/1/delete\">
```

Nếu admin đang login, browser sẽ gửi request delete với cookie của admin.

Laravel bảo vệ bằng CSRF token:

```
<form method=\"POST\" action=\"...\">
    @csrf  <!-- Tạo hidden input với token random -->
    ...
</form>
```

Cách hoạt động:

1. Server tạo random token, lưu trong session
2. Form gửi token này lên server
3. Middleware `VerifyCsrfToken` so sánh token
4. Nếu không khớp → 419 Error

Trang web độc hại không biết token → không thể giả mạo request."

Q3.4: Validation bảo vệ hệ thống như thế nào?

Cách trả lời:

"Validation là **first line of defense** - kiểm tra dữ liệu trước khi xử lý.

Ví dụ trong MovieController:

```
$validated = $request->validate([
    'title' => 'required|string|max:255',
    'rating' => 'nullable|numeric|min:0|max:10',
    'status' => 'required|in:now_showing,coming_soon,ended',
    'poster_url' => 'nullable|url',
    'genres.*' => 'exists:genres,id',  // Kiểm tra FK
]);
```

Các bảo vệ:

- `required`: Không cho submit empty
- `max:255`: Chống Buffer Overflow
- `in:a,b,c`: Chỉ cho phép giá trị hợp lệ
- `url`: Validate format URL
- `exists`: Kiểm tra FK tồn tại trong database

> Nếu validation fail, Laravel tự động:
>
> - Redirect về form cũ
> - Hiển thị error messages
> - Giữ lại old input"

---

4. CÂU HỎI VỀ PERFORMANCE

Q4.1: Làm sao tối ưu hiệu năng trang danh sách với nhiều data?

Cách trả lời:

> "Em áp dụng 4 kỹ thuật:
>
> **1. Pagination thay vì load all:**
>
> ```
> // ❌ Load 10,000 movies
> $movies = Movie::all();
>
> // ☑ Load 20 movies/page
> $movies = Movie::paginate(20);
> ```
>
> **2. Eager Loading:**
>
> ```
> // Load movies + genres trong 2 queries
> $movies = Movie::with('genres')->paginate(20);
> ```
>
> **3. Select only needed columns:**
>
> ```
> // Chỉ lấy cột cần thiết cho danh sách
> $movies = Movie::select('id', 'title', 'poster_url', 'status')
>     ->paginate(20);
> ```
>
> **4. Indexing trong database:**
>
> ```
> -- Index cho các cột hay search/filter
> CREATE INDEX idx_movies_status ON movies(status);
> CREATE INDEX idx_bookings_date ON bookings(created_at);
> ```"

Q4.2: Giải thích cách pagination hoạt động?

Cách trả lời:

> "Laravel Pagination hoạt động như sau:
>
> **Ở Controller:**
>
> ```
> $movies = Movie::latest()->paginate(20);
> // Tự động thêm LIMIT 20 OFFSET (page-1)*20
> ```
>
> **Ở View:**
>
> ```
> @foreach($movies as $movie)
>     {{-- Hiển thị movie --}}
> @endforeach
>
> {{-- Pagination links --}}
> {{ $movies->links() }}
> ```
>
> **Khi user click page 3:**
>
> - URL: `/admin/movies?page=3`
> - Query: `SELECT * FROM movies LIMIT 20 OFFSET 40`
>
> **Để giữ filter khi paginate:**
>
> ```
> $movies = Movie::where('status', $request->status)
>     ->paginate(20)
>     ->withQueryString();  // Giữ ?status=now_showing
> ```"

---

5. CÂU HỎI VỀ BUSINESS LOGIC

Q5.1: Công thức tính giá vé hoạt động như thế nào?

Cách trả lời:

> "Giá vé được tính theo công thức:
>
> ```
> FINAL PRICE = BASE_PRICE + SCREEN_PRICE + PEAK_SURCHARGE
> ```
>
> **1. BASE_PRICE (từ seat_types):**
>
> - Standard: 80,000₫
> - VIP: 120,000₫
> - Couple: 180,000₫

> **2. SCREEN_PRICE (từ screen_types thông qua room):**
>
> - 2D: 0₫
> - 3D: 30,000₫
> - IMAX: 50,000₫
>
> **3. PEAK_SURCHARGE (từ showtime_prices):**
>
> - Admin tự đặt khi tạo suất chiếu
> - Giờ cao điểm (18h-21h) có thể +20,000₫ đến +50,000₫
>
> **Ví dụ:**
>
> ```
> VIP seat + IMAX room + Peak hour (21:00)
> = 120,000 + 50,000 + 30,000 = 200,000₫
> ```
>
> Thiết kế này linh hoạt để:
>
> - Admin điều chỉnh giá theo giờ
> - Áp dụng khuyến mãi
> - Thay đổi giá cơ bản dễ dàng"

Q5.2: Luồng đặt vé và trạng thái Booking?

Cách trả lời:

> "**Booking có 4 trạng thái chính:**
>
> ```
> pending → confirmed → checked → (completed)
>     ↓           ↓
> expired    cancelled
> ```
>
> **Chi tiết:**
>
> 1. **PENDING**: Mới tạo, chờ thanh toán (15 phút)
>   - Ghế chuyển sang `reserved`
>   - Nếu hết 15 phút → tự động `expired`, ghế được giải phóng
> 2. **CONFIRMED**: Đã thanh toán thành công
>   - Ghế chuyển sang `booked`
>   - Gửi QR code cho khách
> 3. **CHECKED**: Đã check-in tại rạp (quét QR)
>   - Cập nhật `qr_status = 'checked'`
>   - Ghi nhận thời gian check-in

> 4. **CANCELLED**: Bị hủy (bởi user hoặc admin)
>
>   - Ghế giải phóng về `available`
>   - Có thể refund nếu đã paid
>
> 5. **EXPIRED**: Quá hạn thanh toán
>
>   - Tự động từ pending sau 15 phút
>   - Ghế giải phóng"

Q5.3: QR Check-in hoạt động như thế nào?

Cách trả lời:

> "**Quy trình QR Check-in:**
>
> **Bước 1: Tạo QR Code (khi booking thành công)**

```
public static function generateQRCode($bookingId, $seatInfo) {
    return hash('sha256', $bookingId . '_' . $seatInfo . '_' . microtime());
}
```

> - Dùng SHA256 hash để tạo mã unique
> - Lưu vào `booking_seats.qr_code`
> - `qr_status = 'active'`
>
> **Bước 2: Admin quét QR tại rạp**

```
public static function checkInWithQR($qrCode) {
    $seats = BookingSeat::where('qr_code', $qrCode)
        ->where('qr_status', 'active')
        ->get();

    if ($seats->isEmpty()) {
        return ['success' => false, 'message' => 'QR không hợp lệ'];
    }

    // Cập nhật trạng thái
    BookingSeat::where('qr_code', $qrCode)->update([
        'qr_status' => 'checked',
        'checked_at' => now()
    ]);

    return ['success' => true, 'message' => 'Check-in thành công'];
}
```

> **An toàn:**

> - Mỗi QR chỉ dùng được 1 lần
> - Sau check-in: `qr_status = 'checked'`
> - Nếu quét lại → báo lỗi "đã sử dụng""

---

6. CÂU HỎI VỀ UX/UI

Q6.1: Tại sao chọn color scheme này cho Admin?

Cách trả lời:

> "Em chọn color scheme với 3 tiêu chí:
>
> **1. Phù hợp branding rạp phim:**
>
> - Deep Teal (#006D77): Tạo cảm giác chuyên nghiệp
> - Burnt Peach (#E29578): Điểm nhấn ấm áp
> - Kết hợp tạo cảm giác như bước vào rạp chiếu phim
>
> **2. Status colors có ý nghĩa:**
>
> - Green (success): Confirmed, Paid, Active
> - Yellow (warning): Pending, Coming Soon
> - Red (danger): Cancelled, Error, Delete
> - Đây là convention người dùng quen thuộc
>
> **3. Seat colors trực quan:**
>
> - Green: Standard (phổ thông, dễ tiếp cận)
> - Gold: VIP (cao cấp, đặc biệt)
> - Pink: Couple (lãng mạn, cho cặp đôi)
>
> Color scheme này đảm bảo **accessibility** với contrast ratio > 4.5:1."

Q6.2: Thiết kế Seat Map dựa trên nguyên tắc gì?

Cách trả lời:

> "Seat Map thiết kế theo **Cinema-like Experience:**
>
> **1. Spatial Awareness:**
>
> - Screen ở trên cùng → user biết hướng màn hình
> - Row labels (A, B, C...) bên trái → dễ định vị
> - Ghế gần screen giá thấp hơn (thực tế)
>
> **2. Visual Hierarchy:**
>
> - Màu phân biệt rõ 3 loại ghế
> - Ghế đang chọn có glow effect nổi bật
> - Legend luôn hiển thị để tham chiếu

> **3. Interaction Feedback:**
>
> - Hover: ghế nâng lên (translateY -3px)
> - Click: màu đổi sang blue (selected)
> - Couple seat: rộng gấp đôi, hiển thị "1-2"
>
> **4. Responsive:**
>
> - Desktop: ghế 45x45px
> - Tablet: ghế 40x40px
> - Mobile: ghế 30x30px
> - Đảm bảo vẫn touch-friendly trên mobile"

Q6.3: Tại sao dùng Sidebar Offcanvas thay vì Modal?

Cách trả lời:

> "Trong trang Edit Seat Type, em dùng Sidebar thay Modal vì:
>
> **1. Context Preservation:**
>
> - Sidebar slide từ phải, seat map vẫn visible (dù mờ)
> - User vẫn thấy ghế mình đang chọn
> - Modal che hết, mất context
>
> **2. Space Efficiency:**
>
> - Sidebar dùng không gian bên phải
> - Seat map vẫn chiếm phần lớn màn hình
> - Phù hợp với workflow: chọn ghế → edit → apply
>
> **3. Consistent with Modern Apps:**
>
> - Gmail, Slack, Notion đều dùng sidebar pattern
> - User quen thuộc với UX này
>
> **4. Keyboard Friendly:**
>
> - Escape để đóng
> - Tab để navigate giữa options
>
> Sidebar cũng full-width trên mobile để đảm bảo usability."

---

7. CÂU HỎI VỀ TESTING & DEPLOYMENT

Q7.1: Bạn test dự án như thế nào?

Cách trả lời:

> "Em test dự án ở nhiều cấp độ:

> **1. Manual Testing:**
>
> - Test từng chức năng CRUD
> - Test các edge cases (empty, max length...)
> - Test flows (booking flow, cancel flow...)
>
> **2. Browser Testing:**
>
> - Chrome, Firefox, Edge, Safari
> - Responsive testing với DevTools
> - Mobile testing trên thiết bị thật
>
> **3. Database Testing:**
>
> - Verify data integrity sau mỗi thao tác
> - Check foreign key constraints
> - Test transaction rollback
>
> **4. Security Testing:**
>
> - Thử truy cập admin routes khi chưa login
> - Thử truy cập với user thường
> - Test SQL injection trong search
>
> *Nếu có thời gian, em sẽ viết automated tests với PHPUnit và Laravel Dusk cho UI testing.*"

Q7.2: Dự án deploy như thế nào?

Cách trả lời:

> "Quy trình deploy cho production:
>
> **1. Chuẩn bị:**
>
> ```
> # Compile assets
> npm run build
>
> # Optimize Laravel
> php artisan config:cache
> php artisan route:cache
> php artisan view:cache
> ```
>
> **2. Environment:**
>
> ```
> APP_ENV=production
> APP_DEBUG=false
> DB_CONNECTION=mysql
> DB_HOST=production-server
> ```

> **3. Server Setup:**
>
> - Apache/Nginx với PHP 8.1+
> - MySQL 8.0+
> - SSL certificate (HTTPS)
>
> **4. Deployment Steps:**
>
> - Push code lên Git
> - Pull trên server
> - Run `composer install --optimize-autoloader --no-dev`
> - Run `php artisan migrate --force`
> - Clear và rebuild caches
>
> Em đề xuất dùng Laravel Forge hoặc Envoyer để automate quy trình này."

---

8. CÂU HỎI VỀ LARAVEL FRAMEWORK

Q8.1: Giải thích Service Container trong Laravel?

Cách trả lời:

> "Service Container là **dependency injection container** của Laravel.
>
> **Ví dụ đơn giản:** Thay vì:
>
> ```
> class AdminController {
>     public function index() {
>         $db = new DatabaseConnection();  // Tight coupling
>         $logger = new Logger();          // Hard to test
>     }
> }
> ```
>
> Với Container:
>
> ```
> class AdminController {
>     public function __construct(
>         private DatabaseConnection $db,   // Injected
>         private Logger $logger            // Injected
>     ) {}
> }
> ```
>
> **Lợi ích:**
>
> 1. **Loose Coupling**: Dễ swap implementation
> 2. **Testability**: Mock dependencies khi test
> 3. **Single Responsibility**: Container quản lý creation

> Laravel tự động resolve dependencies khi:
>
> - Controller được instantiate
> - Method injection trong controller actions
> - Middleware, Jobs, Events..."

Q8.2: Middleware là gì? Giải thích luồng request?

Cách trả lời:

> "Middleware là **filter** cho HTTP requests.
>
> **Luồng request qua middleware:**

```
Request → Middleware 1 → Middleware 2 → Controller → Response
            │                │
            ▼                ▼
         (auth)         (role:admin)
```

> **Ví dụ CheckRole middleware:**

```php
public function handle($request, Closure $next, $role) {
    // Before Controller
    if (Auth::user()->role !== $role) {
        abort(403);
    }

    $response = $next($request);  // Gọi Controller

    // After Controller (có thể modify response)
    return $response;
}
```

> **Trong dự án em có:**
>
> - `auth`: Kiểm tra đăng nhập
> - `role:admin`: Kiểm tra quyền admin
> - `VerifyCsrfToken`: Chống CSRF
>
> Middleware chain giúp code clean, mỗi middleware một nhiệm vụ."

Q8.3: Blade Template Engine hoạt động như thế nào?

Cách trả lời:

> "Blade là template engine của Laravel, compile sang PHP thuần.

Các feature chính em dùng:

1. Layout Inheritance:

```
{{-- layouts/admin.blade.php --}}
<html>
    @yield('content')  <!-- Placeholder -->
</html>

{{-- views/admin/dashboard.blade.php --}}
@extends('layouts.admin')
@section('content')
    <h1>Dashboard</h1>
@endsection
```

2. Data Display (XSS Safe):

```
{{ $movie->title }}          <!-- Escaped -->
{!! $movie->description !!}  <!-- Raw HTML -->
```

3. Control Structures:

```
@if($booking->status == 'confirmed')
    <span class=\"badge bg-success\">Confirmed</span>
@elseif($booking->status == 'pending')
    <span class=\"badge bg-warning\">Pending</span>
@endif

@foreach($movies as $movie)
    {{ $movie->title }}
@endforeach
```

4. Components:

```
@include('admin.partials.sidebar')
@yield('extra-js')
@stack('scripts')
```

Blade được compile lần đầu và cache, nên performance tốt."

---

9. CÂU HỎI TÌNH HUỐNG & XỬ LÝ LỖI

Q9.1: Nếu 2 admin cùng sửa 1 movie, xử lý như thế nào?

Cách trả lời:

> "Đây là vấn đề **Race Condition**. Có thể xử lý bằng:
>
> **1. Optimistic Locking (Khuyên dùng):**

```
// Migration
$table->timestamp('updated_at');

// Update check
$movie = Movie::find($id);
if ($movie->updated_at != $request->original_updated_at) {
    return back()->with('error', 'Data đã được cập nhật bởi người khác');
}
```

> **2. Pessimistic Locking:**

```
DB::transaction(function() use ($id) {
    $movie = Movie::lockForUpdate()->find($id);
    // Update movie
});
```

> **3. UI Solution:**
>
> - Hiển thị "Đang được sửa bởi Admin X"
> - Real-time notification với WebSocket
>
> Trong dự án hiện tại, em dùng **Last Write Wins** vì admin panel ít concurrent users. Nếu cần, em sẽ implement Optimistic Locking."

Q9.2: Booking timeout 15 phút hoạt động như thế nào?

Cách trả lời:

> "Hệ thống timeout xử lý booking pending quá 15 phút:
>
> **Cách 1: Scheduled Task (Khuyên dùng)**

```
// app/Console/Kernel.php
$schedule->command('bookings:expire-pending')
    ->everyMinute();

// Command
Booking::where('status', 'pending')
    ->where('created_at', '<', now()->subMinutes(15))
    ->update(['status' => 'expired']);

// Giải phóng ghế
```

```
ShowtimeSeat::whereIn('booking_id', $expiredBookingIds)
    ->update(['status' => 'available']);
```

Cách 2: Check khi load (Đang dùng)

```
// Khi user vào seat map hoặc payment page
$booking = Booking::find($id);
if ($booking->isPending() && $booking->isExpired()) {
    $booking->expire();  // Cập nhật status và giải phóng ghế
}
```

Em đề xuất dùng cách 1 cho production vì:

- Chạy background, không block user
- Đảm bảo ghế được giải phóng đúng giờ
- Không phụ thuộc vào user access"

Q9.3: Xử lý lỗi khi thanh toán thất bại?

Cách trả lời:

"Quy trình xử lý payment failure:

1. Catch Exception:

```
try {
    $result = PaymentGateway::process($booking);
} catch (PaymentException $e) {
    // Log lỗi
    Log::error('Payment failed', [
        'booking_id' => $booking->id,
        'error' => $e->getMessage()
    ]);

    // Giữ booking ở pending, cho phép retry
    return back()->with('error', 'Thanh toán thất bại. Vui lòng thử lại.');
}
```

2. Graceful Handling:

- Không xóa booking, giữ ghế reserved
- Cho phép user retry trong 15 phút
- Nếu hết 15 phút → expire

3. User Feedback:

- Hiển thị lỗi rõ ràng ("Thẻ bị từ chối", "Số dư không đủ")

> - Hướng dẫn next steps
> - Link liên hệ support
>
> **4. Admin Visibility:**
>
> - Dashboard hiển thị failed payments
> - Alert khi có nhiều failures bất thường
>
> Transaction đảm bảo không có partial updates - hoặc thanh toán thành công hoàn toàn, hoặc không có gì thay đổi."

---

10. DEMO SCRIPT & TIPS THUYẾT TRÌNH

10.1. Demo Script (5 phút)

```
=== PHẦN 1: GIỚI THIỆU (30 giây) ===

"Chào thầy/cô, em xin demo hệ thống Admin Panel của CineBook -
ứng dụng đặt vé xem phim. Hệ thống này cho phép quản lý
toàn bộ operations của rạp chiếu phim."

=== PHẦN 2: LOGIN VÀ DASHBOARD (1 phút) ===

"Đầu tiên, em login với tài khoản admin.
[Thao tác login]

Đây là Dashboard với các thống kê:
- Số user, số phim
- Vé bán hôm nay, doanh thu
- Phim doanh thu cao nhất và thấp nhất
- 10 booking gần nhất

Dashboard sử dụng LEFT JOIN để tính doanh thu,
đảm bảo hiển thị cả phim chưa có doanh thu."

=== PHẦN 3: QUẢN LÝ PHIM (1 phút) ===

"Tiếp theo là quản lý phim.
[Vào Movies]

Em sẽ thêm một phim mới.
[Click Add New, điền form]

Validation đảm bảo dữ liệu hợp lệ.
Genres sử dụng many-to-many relationship.
[Submit và show success message]"

=== PHẦN 4: QUẢN LÝ PHÒNG VÀ GHẾ (1.5 phút) ===

"Đây là tính năng em tâm đắc nhất - quản lý phòng chiếu.
[Vào Rooms → Create New]
```

```
Em tạo phòng 8 hàng, 10 ghế mỗi hàng.
[Click Generate Preview]

Seat map hiển thị real-time.
Em có thể chọn ghế và thay đổi loại.
[Chọn vài ghế, mở sidebar, đổi sang VIP]

Có các template sẵn: VIP Center, Couple Back row.
[Apply Cinema Style template]

Database Transaction đảm bảo khi tạo phòng,
tất cả 80 ghế được tạo cùng lúc hoặc không có gì được tạo."

=== PHẦN 5: QR CHECK-IN (30 giây) ===

"Cuối cùng là hệ thống QR Check-in.
[Vào QR Check-in]

Nhân viên quét mã QR của khách.
[Nhập mã demo]

Hệ thống xác thực và hiển thị thông tin booking.
SHA256 hash đảm bảo mã không thể giả mạo.
Mỗi QR chỉ dùng được một lần."

=== PHẦN 6: KẾT LUẬN (30 giây) ===

"Tóm lại, Admin Panel được xây dựng với:
- MVC pattern, tách biệt rõ ràng
- Phân quyền 2 lớp: Auth + Role
- Database Transactions cho data integrity
- Responsive design, hoạt động trên mobile

Em xin kết thúc phần demo. Mời thầy/cô đặt câu hỏi."
```

10.2. Tips Thuyết Trình Ăn Điểm

```
┌─────────────────────────────────────────────────────────────────┐
│                    TIPS THUYẾT TRÌNH                            │
├─────────────────────────────────────────────────────────────────┤
│                                                                 │
│   ☑ NÊN LÀM:                                                    │
│   ┌─────────────────────────────────────────────────────┐       │
│   │ • Nói chậm, rõ ràng, tự tin                         │       │
│   │ • Demo thực tế, không chỉ nói lý thuyết             │       │
│   │ • Giải thích WHY, không chỉ WHAT                    │       │
│   │ • Chuẩn bị sẵn dữ liệu demo                         │       │
│   │ • Acknowledge limitations, đề xuất improvements     │       │
│   │ • Trả lời thẳng vào câu hỏi                         │       │
│   │ • Nếu không biết, nói "Em sẽ tìm hiểu thêm"         │       │
```

```
│    └──────────────────────────────────────────────────────┘    │
│                                                                │
│    ❌ KHÔNG NÊN:                                               │
│    ┌──────────────────────────────────────────────────────┐    │
│    │ • Đọc slide/code word by word                        │    │
│    │ • Nói quá nhanh vì lo hết giờ                        │    │
│    │ • Bịa câu trả lời khi không chắc                     │    │
│    │ • Tranh luận với giám khảo                           │    │
│    │ • Demo features chưa test kỹ                         │    │
│    │ • Nói "em không có thời gian làm..."                 │    │
│    └──────────────────────────────────────────────────────┘    │
│                                                                │
│    💡 POWER PHRASES:                                           │
│    ┌──────────────────────────────────────────────────────┐    │
│    │ "Em chọn solution A thay vì B vì..."                 │    │
│    │ "Trade-off ở đây là..."                              │    │
│    │ "Nếu có thêm thời gian, em sẽ..."                    │    │
│    │ "Theo best practices của Laravel..."                 │    │
│    │ "Design pattern em áp dụng là..."                    │    │
│    │ "Database constraint đảm bảo..."                    │    │
│    └──────────────────────────────────────────────────────┘    │
│                                                                │
└────────────────────────────────────────────────────────────────┘
```

10.3. Checklist Trước Demo

```
□ Login credentials hoạt động
□ Database có sample data đẹp
□ Tất cả CRUD operations test xong
□ Không có console errors
□ Internet connection ổn định (nếu dùng CDN)
□ Browser zoom ở mức dễ nhìn
□ Đóng các tabs không liên quan
□ Tắt notifications
□ Backup plan nếu demo fail
□ Biết sẵn 2-3 điểm nổi bật để nhấn mạnh
```

TỔNG KẾT SERIES

Các file trong series:

File	Nội dung
PART 1	Tổng quan kiến trúc, MVC, Routes, Middleware
PART 2	Database, Eloquent, Transactions, Algorithms
PART 3	UX/UI Design, Color Scheme, Components

File	Nội dung
PART 4	Q&A, Demo Script, Tips thuyết trình

Key Takeaways:

1. **Kiến trúc**: MVC + Middleware tách biệt admin/frontend
2. **Database**: Relationships, Transactions, Eager Loading
3. **Security**: Auth + Role + CSRF + Validation
4. **UX/UI**: Cinema-like experience, Status colors, Responsive
5. **Demo**: Tập trung vào WHY, không chỉ WHAT

Chúc bạn thuyết trình thành công! 🎬🎟️