

CINEBOOK ADMIN PANEL - TECHNICAL DOCUMENTATION

Tài liệu kỹ thuật phân tích chi tiết về hệ thống Admin Panel của CineBook

Bao gồm: Design Patterns, Algorithms, Database Optimization, và Ý nghĩa Nghề nghiệp vụ

Phiên bản: 2.0 (Technical Deep Dive Edition) **Ngày cập nhật:** 2026-01-29 **Audience:** Developers, Technical Architects, System Analysts

MỤC LỤC

1. Tổng Quan Hệ Thống
2. Kiến Trúc & Cấu Trúc
3. Design Patterns & Kỹ Thuật Lập Trình
4. Database Architecture & Optimization
5. Algorithms & Performance Analysis
6. Các Module Chính
7. Logic Nghề nghiệp Vụ
8. Tính Năng Realtime & Automation
9. Bảo Mật & Authentication
10. Business Intelligence & Analytics
11. Quy Trình Hoạt Động
12. Ý Nghĩa Thực Tế Trong Vận Hành Rạp Phim
13. Tổng Kết Kỹ Thuật Lập Trình

⌚ Ý NGHĨA THỰC TẾ - TẠI SAO CINEBOOK THIẾT KẾ NHƯ VẬY?

Phần này giải thích TẠI SAO mỗi thông kê và chức năng được thiết kế như vậy, dựa trên cách vận hành THỰC TẾ của các rạp chiếu phim như CGV, Lotte Cinema, Galaxy.

🎬 SO SÁNH VỚI RẠP PHIM THỰC TẾ

Chức năng CineBook	Tương đương Rạp Thực Tế	Ý Nghĩa Business
Vé Bán Hôm Nay (completed showtimes)	Báo cáo doanh thu cuối ca	Chỉ đếm vé đã sử dụng, không đếm vé chưa check-in
Doanh Thu Hôm Nay (completed showtimes)	Doanh thu thực hiện (Earned Revenue)	Tuân thủ nguyên tắc kế toán Revenue Recognition
Doanh Thu Có Thể Mất 24h	Quỹ dự phòng hoàn tiền	Worst-case scenario nếu phải hủy tất cả suất chiếu
Tiền Hoàn Trong Tháng	KPI chất lượng dịch vụ	Monitor tỷ lệ hoàn/hủy để cải thiện

Chức năng CineBook	Tương đương Rạp Thực Tế	Ý Nghĩa Business
Phim Hot Nhất	Xếp hạng Box Office	Quyết định tăng/giảm suất chiếu
Suất Chiếu Đóng Nhất	Peak time analysis	Phân bổ nhân sự bán bắp, soát vé
Đặt Vé Gần Đây	Màn hình POS quầy vé	Realtime monitoring, phát hiện gian lận
Hệ thống màu trạng thái	Đèn tín hiệu giao thông	UX standard toàn cầu

#[1] Ý NGHĨA CỦA TỪNG THỐNG KÊ DASHBOARD

① VÉ BÁN HÔM NAY (Tickets Sold Today)

? Tại sao chỉ đếm vé của suất chiếu ĐÃ KẾT THÚC?

Hãy tưởng tượng bạn là quản lý CGV Vincom. Bây giờ là 3 giờ chiều.

- Vé suất đã kết thúc = **100% doanh thu chắc chắn** (khách đã xem)
- Vé suất đang/chưa chiếu = **Có thể hoàn tiền** (khách chưa đến, sự cố...)

Trong thực tế rạp phim:

- Vé suất đã kết thúc = **100% doanh thu chắc chắn** (khách đã xem)
- Vé suất đang/chưa chiếu = **Có thể hoàn tiền** (khách chưa đến, sự cố...)

Ứng dụng thực tế:

- Báo cáo cho sếp/chủ rạp về số vé THỰC SỰ đã phục vụ
- So sánh với ngày hôm qua cùng khung giờ
- Đánh giá hiệu suất nhân viên theo ca

② DOANH THU HÔM NAY (Revenue Today)

? Tại sao chỉ tính doanh thu từ suất chiếu đã kết thúc?

Theo nguyên tắc kế toán **Revenue Recognition** (Ghi nhận doanh thu):

- Trước khi chiếu: Tiền khách trả là **Doanh thu chưa thực hiện** (Unearned Revenue)
- Sau khi chiếu: Tiền đó mới là **Doanh thu thực hiện** (Earned Revenue)

Ví dụ thực tế:

Khách A mua vé 100.000đ cho suất 20:00 tối nay

- └─ 15:00: Khách đặt vé → Rạp CHƯA được ghi nhận doanh thu
- └─ 20:00: Phim bắt đầu → Vẫn chưa (khách có thể bỏ về)
- └─ 22:00: Phim kết thúc → BÂY GIỜ mới tính là doanh thu

Ứng dụng thực tế:

- Báo cáo tài chính chính xác cho kế toán
- So sánh doanh thu thực tế giữa các ngày
- Tránh overestimate (ước tính quá cao) doanh thu

③ DOANH THU CÓ THỂ MẤT 24H (Revenue at Risk)

? Metric này có ý nghĩa gì trong thực tế?

"Doanh Thu Có Thể Mất" = Tổng tiền từ vé đã bán cho suất chiếu trong 24h tới. Nếu có sự cố (thiên tai, dịch bệnh, mất điện), đây là số tiền TỐI ĐA phải hoàn trả.

Ví dụ thực tế:

Thời điểm: 10:00 sáng 29/01/2026
Dashboard hiện: "Doanh Thu Có Thể Mất: 50.000.000đ"

Điều này nghĩa là:

- └─ Trong 24h tới có nhiều suất chiếu
- └─ Tổng vé đã bán = 50 triệu
- └─ Nếu phải hủy tất cả → Hoàn tối đa 50 triệu
- └─ Cần chuẩn bị quỹ dự phòng ít nhất 50 triệu

Ứng dụng thực tế:

- **Quản trị rủi ro tài chính:** Biết worst-case scenario
- **Chuẩn bị quỹ dự phòng:** Đảm bảo có tiền hoàn nếu cần
- **Quyết định kinh doanh:** Con số cao = booking tốt, con số thấp = cần marketing

④ TIỀN HOÀN TRONG THÁNG (Refund Amount This Month)

? Tại sao cần theo dõi metric này?

Refund rate là KPI quan trọng đo lường CHẤT LƯỢNG DỊCH VỤ của rạp.

Phân tích theo tình huống:

Refund Amount	Ý nghĩa	Hành động
Thấp (<5% doanh thu)	Dịch vụ tốt, ít khiếu nại	Duy trì chất lượng
Trung bình (5-10%)	Bình thường	Theo dõi trend
Cao (>10%)	Có vấn đề!	Điều tra nguyên nhân

Nguyên nhân refund cao có thể là:

- 🎟 Sự cố kỹ thuật (máy chiếu hỏng, âm thanh lỗi)

- 🎬 Phim chất lượng kém (khách đòi hoàn tiền)
- 🕒 Thời tiết xấu (khách không đến được)
- 📱 UX website kém (khách đặt nhầm)

Ứng dụng thực tế:

- So sánh refund tháng này vs tháng trước
- Điều tra đột biến refund
- Cải thiện quy trình để giảm refund

5 PHIM HOT NHẤT (Top Movie by Revenue)

? Thông tin này giúp gì cho vận hành?

Trong thực tế, CGV/Lotte dùng data này để:

Quyết định lịch chiếu:

- Phim A: Doanh thu cao → Tăng suất chiếu
Phim B: Doanh thu thấp → Giảm suất, chuyển sang phòng nhỏ
Phim C: Doanh thu cực thấp → Rút khỏi lịch chiếu

Đàm phán với nhà phát hành:

- Phim hot → Rạp có leverage yêu cầu ưu đãi
- Data doanh thu → Chứng minh performance

Marketing:

- Promote phim hot trên homepage
- Bundle deals cho phim ế

6 SUẤT CHIẾU ĐÔNG NHẤT (Top Showtime Today)

? Thông tin này dùng để làm gì?

Đây là thông tin VÀNG cho team vận hành hàng ngày.

Phân bổ nhân sự:

- Suất 18:00 có 90% ghế → Peak hour!

 - └─ 17:30: Tăng nhân viên bán bắp
 - └─ 17:45: Tăng nhân viên soát vé
 - └─ 18:00: Chuẩn bị check-in nhanh
 - └─ 20:30: Tăng nhân viên dọn phòng

Chuẩn bị logistics:

- Parking đỗ → Bố trí bảo vệ
 - Lobby đỗ → Mở thêm quầy
 - VIP room đỗ → Chuẩn bị dịch vụ đặc biệt
-

Ý NGHĨA CỦA CÁC CHỨC NĂNG QUẢN LÝ

QUẢN LÝ PHIM (Movies Management)

Tại sao KHÔNG cho xóa phim?

Xóa phim = Mất toàn bộ lịch sử booking, review, doanh thu của phim đó. Thay vào đó: Đặt status = "ended" để archive.

Tại sao auto-update status coming_soon → now_showing?

Tránh quên update thủ công khi phim ra mắt. Tự động chạy khi release_date đến.

Tại sao không cho end phim khi còn suất chiếu tương lai?

Tránh conflict: Phim ended nhưng vẫn có lịch chiếu. Phải cancel/xóa hết suất chiếu trước khi end.

QUẢN LÝ PHÒNG & GHẾ (Rooms & Seats)

Công thức tính giá vé:

$$\begin{aligned}\text{Giá vé} &= \text{Giá loại màn hình} + \text{Giá loại ghế} \\ &= \text{Screen Type Price} + \text{Seat Type Price}\end{aligned}$$

Ví dụ:

- └─ IMAX (screen): 50.000đ
- └─ VIP (seat): 30.000đ
- └─ Tổng: 80.000đ

Tại sao ghế Couple luôn theo cặp?

Ghế Couple là ghế đôi (sofa) cho cặp đôi. 1 booking Couple = 2 ghế liền nhau = 1 QR code = 1 giá (không x2).

QUẢN LÝ SUẤT CHIẾU (Showtimes)

Tại sao cần Overlap Detection?

Room A: |----Avatar 14:00-16:30----|
 |----Dune 15:00-17:30----| ← KHÔNG THỂ! 

1 phòng không thể chiếu 2 phim cùng lúc.
System tự động detect và block.

Tại sao chỉ cho edit PRICING của showtime?

Nếu đã có booking, việc đổi movie/room/time sẽ:

- Khách đã đặt vé cho phim A, đổi sang phim B → Chaos!
- Khách đặt Room 1, đổi sang Room 2 → Ghế đã chọn không còn! → Chỉ cho edit giá vé (không ảnh hưởng booking đã có).

Empty Showtime Filter dùng để làm gì?

Tìm suất chiếu không có ai đặt → Cần marketing/promotion. Hoặc cancel để tiết kiệm chi phí vận hành.

☒ QUẢN LÝ BOOKING (Bookings)

Flow hủy booking của Admin:

- ```

Admin Cancel
↓
1. Kiểm tra: Suất chiếu chưa kết thúc?
↓
2. Update booking.status = 'cancelled'
↓
3. Release tất cả ghế → available
↓
4. Gửi email thông báo cho khách
↓
5. Ghi nhận refund amount

```

## Tại sao cần track refund?

*Refund = Chi phí vận hành. Cần minimize. High refund rate → Có vấn đề cần fix.*

## ☒ QR CHECK-IN SYSTEM

### Flow check-in tại rạp thực tế:

- ```

Khách đến rạp
↓
Đưa QR code (trên điện thoại)
↓
Nhân viên scan bằng tablet/máy quét
↓
System validate:
|— QR hợp lệ?
|— Booking đã thanh toán?
|— Chưa check-in trước đó?

```

```
|— Suất chiếu chưa kết thúc?  
|  ↓  
|  Hiện thông tin: Tên khách, Phim, Ghế  
|  ↓  
Khách vào phòng chiếu
```

Tại sao cần QR status?

Status	Ý nghĩa
active	Chưa dùng, có thể check-in
used	Đã check-in, không dùng lại được
expired	Suất chiếu đã qua, vé vô hiệu

Ý NGHĨA CỦA EMAIL AUTOMATION

1. Email Nhắc Nhở (Showtime Reminder)

Gửi trước 2 tiếng:

Lý do: Khách cần thời gian chuẩn bị, di chuyển đến rạp. Nội dung: Phim gì, Giờ nào, Phòng nào, Ghế nào.

Tác dụng thực tế:

- ⬇️ Giảm no-show rate (khách quên không đến)
- ⬆️ Tăng customer satisfaction
- 📊 Track được ai đã nhận reminder

2. Email Yêu Cầu Review

Gửi sau 2 tiếng khi phim kết thúc:

Lý do: Khách đã về nhà, có thời gian suy nghĩ về phim. Không gửi ngay khi ra rạp → Spam, khách chưa sẵn sàng.

Tác dụng thực tế:

- ⬆️ Tăng số lượng review
- ⬆️ Tăng chất lượng review (khách có thời gian viết)
- 📊 Social proof cho khách hàng mới

Ý NGHĨA CỦA BẢO MẬT

Role-Based Access Control

Tại sao cần phân quyền Admin/User?

User thường: Đặt vé, Xem phim, Viết review
Admin: Quản lý phim, Suất chiếu, Hủy booking, Xem doanh thu

→ Tách biệt để bảo vệ dữ liệu nhạy cảm
→ User không thể xem doanh thu, hủy booking người khác

Tại sao Admin không tự đổi role của mình?

Tránh tình huống: Admin duy nhất tự đổi thành User → Không ai quản lý được.

1. TỔNG QUAN HỆ THỐNG

1.1 Giới Thiệu

CineBook Admin Panel là hệ thống quản lý toàn diện cho rạp chiếu phim, được xây dựng trên Laravel Framework. Hệ thống cung cấp các công cụ quản lý từ cơ bản đến nâng cao cho việc vận hành rạp phim.

1.2 Mục Đích

- Quản lý vận hành:** Movies, Rooms, Showtimes, Bookings
- Quản lý người dùng:** Users, Reviews, Roles
- Business Analytics:** Dashboard với metrics thời gian thực
- Automation:** Email marketing, Reminders, Review requests
- Check-in Management:** QR Code scanning và validation

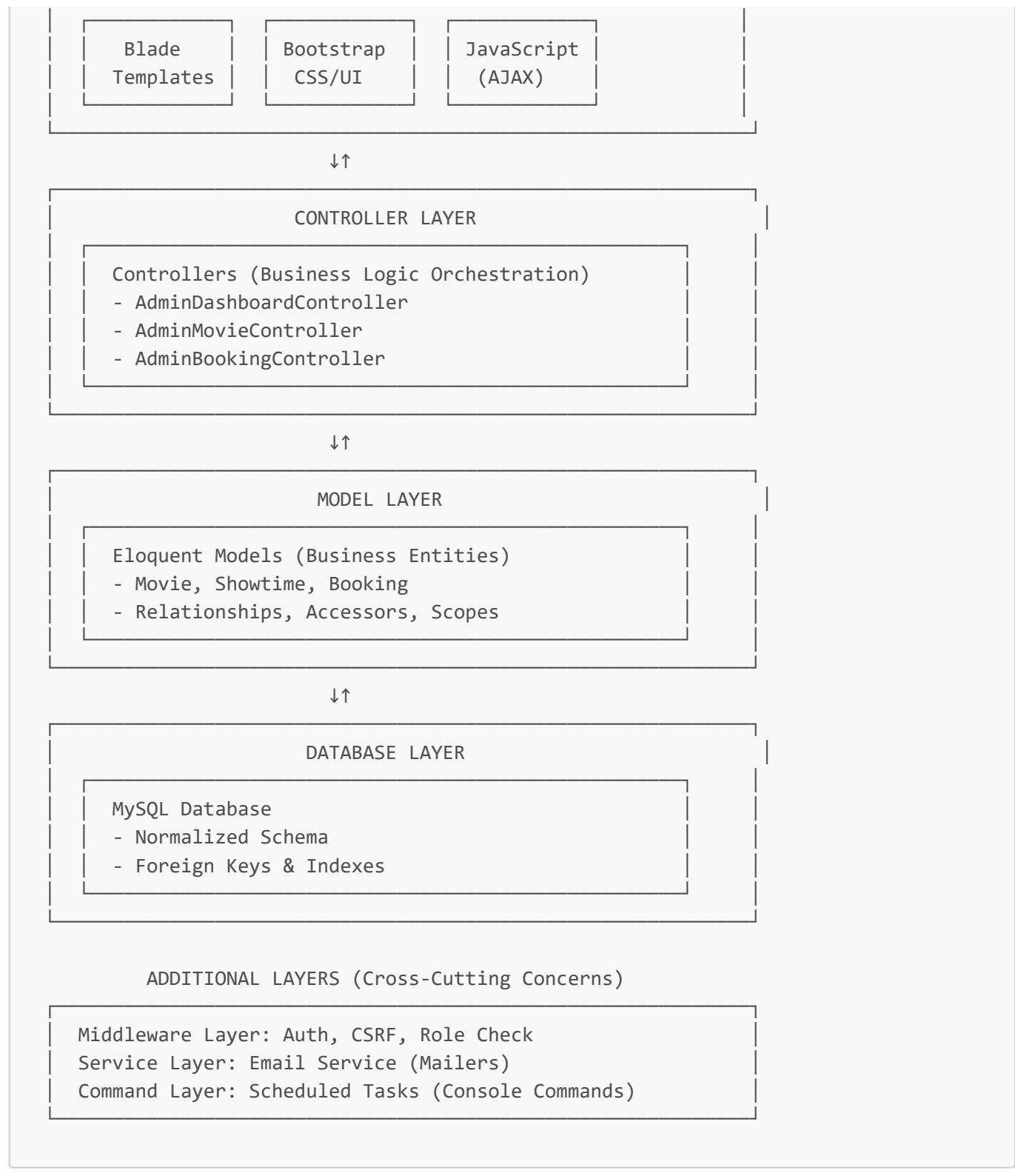
1.3 Công Nghệ Sử Dụng

Technology	Version	Purpose
Laravel	11.x	Backend Framework
PHP	8.2+	Programming Language
MySQL	8.0+	Database
Blade	Latest	Template Engine
Bootstrap	5.x	UI Framework
Carbon	Latest	Date/Time Manipulation

2. KIẾN TRÚC & CẤU TRÚC

2.0 Overview Kiến Trúc Tổng Thể

CineBook được xây dựng theo kiến trúc **MVC (Model-View-Controller)** của Laravel với các layer bổ sung:



3. DESIGN PATTERNS & KỸ THUẬT LẬP TRÌNH

3.1 🎨 Design Patterns Được Sử Dụng

3.1.1 Active Record Pattern (Eloquent ORM)

Mô tả:

- Mỗi Model class đại diện cho 1 table trong database
- Model object = 1 row trong table

- Methods trên Model object = operations trên database

Ví dụ Implementation:

```
// Model: app/Models/Booking.php
class Booking extends Model
{
    protected $fillable = ['user_id', 'showtime_id', 'total_price', 'status'];

    // Relationships (Active Record Pattern)
    public function user() {
        return $this->belongsTo(User::class);
    }

    public function showtime() {
        return $this->belongsTo(Showtime::class);
    }
}

// Usage in Controller
$booking = Booking::find(1);           // SELECT * FROM bookings WHERE id = 1
$booking->status = 'confirmed';       // Object manipulation
$booking->save();                     // UPDATE bookings SET status = ...

```

Ý nghĩa nghiệp vụ:

- **Simplicity:** Code đơn giản, dễ đọc
- **Productivity:** Không cần viết SQL thủ công
- **Maintainability:** Business logic tập trung trong Model
- **Trade-off:** Performance có thể chậm hơn raw SQL cho complex queries

3.1.2 Accessor & Mutator Pattern

Mô tả:

- **Accessor:** Tự động transform data khi READ từ database
- **Mutator:** Tự động transform data khi WRITE vào database

Ví dụ Implementation:

```
// Model: app/Models>Showtime.php
class Showtime extends Model
{
    /**
     * ACCESSOR: Tính toán start_datetime từ show_date + show_time
     * Được gọi khi: $showtime->start_datetime
     */
    public function getStartDatetimeAttribute(): Carbon
    {

```

```

$date = Carbon::parse($this->show_date);
$time = Carbon::parse($this->show_time);

return $date->copy()->setTimeFrom($time);
}

/**
 * ACCESSOR: Tính toán end_datetime = start + duration
 * Được gọi khi: $showtime->end_datetime
 */
public function getEndDatetimeAttribute(): Carbon
{
    $duration = $this->movie->duration ?? 0;
    return $this->start_datetime->copy()->addMinutes($duration);
}

/**
 * ACCESSOR: Tự động tính status dựa trên thời gian hiện tại
 * Được gọi khi: $showtime->status
 */
public function getStatusAttribute(): string
{
    $now = Carbon::now();

    if ($now->gt($this->end_datetime)) {
        return 'done';           // Đã kết thúc
    }

    if ($now->gte($this->start_datetime) && $now->lte($this->end_datetime)) {
        return 'ongoing';       // Đang chiếu
    }

    return 'upcoming';         // Chưa chiếu
}
}

```

Kỹ thuật lập trình:

1. Lazy Evaluation:

- **status** không được lưu trong database
- Được tính toán real-time mỗi lần access
- Đảm bảo luôn chính xác với thời gian hiện tại

2. Separation of Concerns:

- Business logic (tính status) nằm trong Model
- Controller không cần biết cách tính status
- View chỉ cần gọi **\$showtime->status**

Ý nghĩa nghiệp vụ:

```
// Trong Controller:
$showtimes = Showtime::all();

// Trong View:
@foreach($showtimes as $showtime)
    <span class="badge {{ $showtime->status_class }}">
        {{ $showtime->status }} <!-- Tự động hiện: upcoming/ongoing/done -->
    </span>
@endforeach

// ✅ Không cần if/else phức tạp
// ✅ Status luôn chính xác theo thời gian thực
// ✅ Code maintainable, dễ test
```

Performance Consideration:

```
// ⚠️ N+1 Query Problem nếu không cẩn thận
$showtimes = Showtime::all(); // 1 query

foreach ($showtimes as $showtime) {
    echo $showtime->end_datetime; // Gọi $showtime->movie->duration
    // → Query movie cho mỗi showtime! (N queries)
}

// ✅ Solution: Eager Loading
$showtimes = Showtime::with('movie')->all(); // 2 queries total
foreach ($showtimes as $showtime) {
    echo $showtime->end_datetime; // No additional query
}
```

3.1.3 Query Scope Pattern

Mô tả:

- Định nghĩa reusable query constraints trong Model
- Tái sử dụng logic query phức tạp ở nhiều nơi

Ví dụ Implementation:

```
// Model: app/Models>Showtime.php
class Showtime extends Model
{
    /**
     * Scope: Lấy các showtimes sắp chiếu
     */
    public function scopeUpcoming($query)
    {
```

```

        return $query->whereRaw(
            "CONCAT(show_date, ' ', show_time) > ?",
            [Carbon::now()])
    );
}

/***
 * Scope: Lấy các showtimes đã kết thúc
 */
public function scopeEnded($query)
{
    return $query->whereHas('movie', function ($q) {
        $q->whereRaw(
            "DATE_ADD(CONCAT(showtimes.show_date, ' ', showtimes.show_time),
            INTERVAL movies.duration MINUTE) < ?",
            [Carbon::now()])
    );
});

/***
 * Scope: Lấy showtimes đang hoạt động (upcoming hoặc ongoing)
 */
public function scopeActive($query)
{
    return $query->whereHas('movie', function ($q) {
        $q->whereRaw(
            "DATE_ADD(CONCAT(showtimes.show_date, ' ', showtimes.show_time),
            INTERVAL movies.duration MINUTE) > ?",
            [Carbon::now()])
    );
});
}

```

Usage trong Controller:

```

// Thay vì viết raw query mỗi lần:
$upcoming = Showtime::whereRaw("CONCAT(show_date, ' ', show_time) > ?",
[Carbon::now()])
->get();

// Sử dụng Scope - Clean & Readable:
$upcoming = Showtime::upcoming()->get();
$ended = Showtime::ended()->get();
$active = Showtime::active()->get();

// Chain multiple scopes:
$activeInRoom1 = Showtime::active()
->where('room_id', 1)
->orderBy('show_date')
->get();

```

Kỹ thuật lập trình:

1. DRY Principle (Don't Repeat Yourself):

- Query logic được define 1 lần duy nhất
- Reuse ở mọi nơi cần
- Thay đổi logic chỉ cần sửa 1 chỗ

2. Fluent Interface:

- Scopes có thể chain với nhau
- Readable như natural language

Ý nghĩa nghiệp vụ:

```
// Dashboard Controller
$activeShowtimes = Showtime::active()->count();

// Booking Controller
$upcomingShowtimes = Showtime::upcoming()
    ->where('movie_id', $movieId)
    ->get();

// Admin Controller
$endedShowtimes = Showtime::ended()
    ->whereDate('show_date', $today)
    ->with('movie', 'room')
    ->get();

//  Code tự document (self-documenting)
//  Consistent logic across codebase
//  Easy to test and maintain
```

3.1.4 Repository Pattern (Implicit trong Eloquent)

Mô tả:

- Eloquent Models hoạt động như Repositories
- Cung cấp interface để access database
- Abstraction layer giữa business logic và data layer

Ví dụ:

```
// Eloquent Model = Repository
class Booking extends Model
{
    // Repository methods
```

```

public static function findByQRCode($qrCode) {
    return self::whereHas('bookingSeats', function($q) use ($qrCode) {
        $q->where('qr_code', $qrCode);
    })->first();
}

public static function getTodayRevenue() {
    return self::where('payment_status', 'paid')
        ->whereDate('created_at', Carbon::today())
        ->sum('total_price');
}
}

// Usage
$booking = Booking::findByQRCode('ABC123');
$revenue = Booking::getTodayRevenue();

```

Ý nghĩa:

- Data access logic nằm trong Model/Repository
- Controller chỉ gọi methods, không cần biết SQL
- Dễ dàng switch database hoặc caching layer

3.1.5 Observer Pattern (Laravel Events - Potential Enhancement)

Mô tả:

- Tự động trigger actions khi Model events xảy ra
- Decoupling: Event listeners không biết về nhau

Ví dụ (chưa implement, recommended):

```

// app/Observers/BookingObserver.php
class BookingObserver
{
    /**
     * Khi booking được created và paid
     */
    public function created(Booking $booking)
    {
        if ($booking->payment_status === 'paid') {
            // Tự động gửi confirmation email
            Mail::to($booking->user->email)
                ->send(new BookingConfirmationMail($booking));
        }
    }

    /**
     * Khi booking bị cancelled
     */
}

```

```

public function updated(Booking $booking)
{
    if ($booking->isDirty('status') && $booking->status === 'cancelled') {
        // Tự động release seats
        $this->releaseSeats($booking);

        // Gửi cancellation email
        Mail::to($booking->user->email)
            ->send(new BookingCancellationMail($booking));
    }
}

// Register in AppServiceProvider
Booking::observe(BookingObserver::class);

```

Benefits:

- Tự động hóa side effects
 - Maintainable code
 - Dễ test
-

3.1.6 Strategy Pattern (Pricing Strategy)**Mô tả:**

- Khác nhau pricing strategies có thể swap được
- Peak hour pricing, membership pricing, etc.

Current Implementation:

```

// Showtime-specific pricing
class ShowtimePrice extends Model
{
    // Base price
    protected $fillable = ['showtime_id', 'seat_type_id', 'price'];
}

// Final price calculation strategy
class Booking {
    public function calculateTotalPrice($seats, $showtime) {
        $total = 0;

        foreach ($seats as $seat) {
            // Strategy: Screen Price + Seat Type Price + Showtime Modifier
            $screenPrice = $showtime->room->screenType->base_price;
            $seatPrice = $seat->seatType->base_price;
            $showtimeModifier = $showtime->getSeatTypePrice($seat->seat_type_id);

            $total += $screenPrice + $showtimeModifier;
        }
    }
}

```

```

        }

        return $total;
    }
}

```

Ý nghĩa nghiệp vụ:

- Flexible pricing (peak hours, special events)
- Có thể thêm discount strategies sau này
- A/B testing pricing models

3.2 Kỹ Thuật Lập Trình Nâng Cao

3.2.1 Eager Loading vs Lazy Loading

Problem: N+1 Query

```

// ✗ BAD: N+1 Query Problem
$bookings = Booking::all(); // 1 query

foreach ($bookings as $booking) {
    echo $booking->user->name; // N queries (1 per booking)
    echo $booking->showtime->movie->title; // N queries
}
// Total: 1 + N + N = 1 + 2N queries (for 100 bookings = 201 queries!)

```

Solution: Eager Loading

```

// ✓ GOOD: Eager Loading
$bookings = Booking::with(['user', 'showtime.movie'])->all(); // 3 queries total

foreach ($bookings as $booking) {
    echo $booking->user->name; // No query
    echo $booking->showtime->movie->title; // No query
}
// Total: 3 queries only!

```

Kỹ thuật:

```

// Multiple levels of relationships
Booking::with([
    'user', // Load user
    'showtime' => function($query) {
        $query->with(['movie', 'room']); // Load movie and room
    },
]

```

```
'bookingSeats.seat'           // Load booking seats và seats
])->get();

// Conditional eager loading
$bookings = Booking::with([
    'user',
    'showtime' => function($query) use ($includeMovie) {
        if ($includeMovie) {
            $query->with('movie');
        }
    }
])->get();
```

3.2.2 Query Optimization Techniques

1. Select Only Needed Columns

```
// ✗ BAD: Load all columns
$users = User::all(); // SELECT * FROM users

// ✓ GOOD: Select specific columns
$users = User::select('id', 'name', 'email')->get();
// SELECT id, name, email FROM users
```

2. Use Exists Instead of Count

```
// ✗ BAD: Count all rows
if (Booking::where('user_id', $userId)->count() > 0) {
    // User has bookings
}

// ✓ GOOD: Check existence only
if (Booking::where('user_id', $userId)->exists()) {
    // Stop after finding first match
}
```

3. Chunk Large Datasets

```
// ✗ BAD: Load all in memory
$bookings = Booking::all(); // 100,000 bookings = Memory overflow!

foreach ($bookings as $booking) {
    // Process
}

// ✓ GOOD: Process in chunks
```

```
Booking::chunk(1000, function($bookings) {
    foreach ($bookings as $booking) {
        // Process 1000 at a time
    }
});
```

4. Use Raw Expressions for Complex Calculations

```
// ✗ BAD: Load all then calculate in PHP
$movies = Movie::with('showtimes.bookings')->get();

foreach ($movies as $movie) {
    $revenue = 0;
    foreach ($movie->showtimes as $showtime) {
        foreach ($showtime->bookings as $booking) {
            if ($booking->payment_status === 'paid') {
                $revenue += $booking->total_price;
            }
        }
    }
    $movie->revenue = $revenue;
}

// ✓ GOOD: Calculate in database
$movies = Movie::leftJoin('showtimes', 'movies.id', '=', 'showtimes.movie_id')
    ->leftJoin('bookings', function($join) {
        $join->on('showtimes.id', '=', 'bookings.showtime_id')
            ->where('bookings.payment_status', 'paid');
    })
    ->select('movies.*', DB::raw('COALESCE(SUM(bookings.total_price), 0) as revenue'))
    ->groupBy('movies.id')
    ->get();
```

Ý nghĩa:

- Database engine được optimize để tính toán aggregate functions
- Giảm memory usage trong PHP
- Faster execution

3.2.3 Transaction Management

ACID Properties:

- **Atomicity:** All or nothing
- **Consistency:** Database constraints maintained
- **Isolation:** Concurrent transactions don't interfere
- **Durability:** Committed data persists

Implementation:

```
// Booking cancellation với transaction
public function cancel(Booking $booking)
{
    DB::beginTransaction();

    try {
        // Step 1: Update booking status
        $booking->update(['status' => 'cancelled']);

        // Step 2: Release all seats
        foreach ($booking->bookingSeats as $bookingSeat) {
            DB::table('showtime_seats')
                ->where('showtime_id', $booking->showtime_id)
                ->where('seat_id', $bookingSeat->seat_id)
                ->update([
                    'status' => 'available',
                    'reserved_until' => null,
                    'reserved_by_user_id' => null
                ]);
        }

        // Step 3: Record refund
        $booking->update(['refund_amount' => $booking->total_price]);

        // All steps succeeded → Commit
        DB::commit();

        // Step 4: Send email (outside transaction)
        Mail::to($booking->user->email)
            ->send(new BookingCancellationMail($booking));

        return true;
    } catch (\Exception $e) {
        // Any step failed → Rollback all changes
        DB::rollBack();

        Log::error('Booking cancellation failed', [
            'booking_id' => $booking->id,
            'error' => $e->getMessage()
        ]);

        return false;
    }
}
```

Kỹ thuật:**1. Critical Operations Inside Transaction:**

- Database updates
- Data consistency operations

2. Non-Critical Operations Outside Transaction:

- Email sending (không cần rollback nếu fail)
- Logging
- File operations

3. Error Handling:

- Catch exceptions
- Rollback on failure
- Log errors for debugging

Ý nghĩa nghiệp vụ:

Scenario: Cancel booking có 3 ghế

Without Transaction:

- └ Update booking status
- └ Release seat 1
- └ Release seat 2
- └ Release seat 3 (Database error!)
- └ Result: Data inconsistent! Booking cancelled nhưng 1 ghế vẫn bị lock

With Transaction:

- └ Update booking status
- └ Release seat 1
- └ Release seat 2
- └ Release seat 3 (Database error!)
- └ Rollback → All changes reverted → Data consistent!

4. DATABASE ARCHITECTURE & OPTIMIZATION

4.1 Database Schema Design

4.1.1 Normalized Schema (3NF - Third Normal Form)

CineBook database tuân thủ Third Normal Form để:

- Eliminate data redundancy
- Ensure data integrity
- Optimize for updates

Core Tables:

```
-- MOVIES TABLE
CREATE TABLE movies (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(255) NOT NULL,
    release_date DATE NOT NULL,
    duration INT NOT NULL,           -- phút
    status ENUM('coming_soon', 'now_showing', 'ended'),
    poster_url TEXT,
    created_at TIMESTAMP,
    updated_at TIMESTAMP,

    -- Indexes for performance
    INDEX idx_status (status),
    INDEX idx_release_date (release_date)
);

-- ROOMS TABLE
CREATE TABLE rooms (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(255) NOT NULL,
    screen_type_id BIGINT,
    total_rows INT,
    seats_per_row INT,

    FOREIGN KEY (screen_type_id) REFERENCES screen_types(id)
);

-- SEATS TABLE (Normalized - không lưu duplicate data)
CREATE TABLE seats (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    room_id BIGINT NOT NULL,
    seat_type_id BIGINT NOT NULL,
    seat_row VARCHAR(10) NOT NULL,     -- A, B, C
    seat_number INT NOT NULL,         -- 1, 2, 3
    seat_code VARCHAR(20) NOT NULL,    -- A1, A2, B3

    FOREIGN KEY (room_id) REFERENCES rooms(id) ON DELETE CASCADE,
    FOREIGN KEY (seat_type_id) REFERENCES seat_types(id),

    -- Unique constraint: 1 phòng không có 2 ghế trùng code
    UNIQUE KEY unique_seat_per_room (room_id, seat_code),

    -- Composite index for fast lookups
    INDEX idx_room_seat (room_id, seat_code)
);

-- SHOWTIMES TABLE
CREATE TABLE showtimes (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    movie_id BIGINT NOT NULL,
    room_id BIGINT NOT NULL,
    show_date DATE NOT NULL,
    show_time TIME NOT NULL,
```

```
FOREIGN KEY (movie_id) REFERENCES movies(id) ON DELETE CASCADE,
FOREIGN KEY (room_id) REFERENCES rooms(id) ON DELETE CASCADE,

-- Index for overlap detection queries
INDEX idx_room_datetime (room_id, show_date, show_time),
INDEX idx_show_date (show_date)
);

-- SHOWTIME_SEATS TABLE (Many-to-Many với status)
CREATE TABLE showtime_seats (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    showtime_id BIGINT NOT NULL,
    seat_id BIGINT NOT NULL,
    status ENUM('available', 'reserved', 'booked') DEFAULT 'available',
    reserved_until TIMESTAMP NULL,
    reserved_by_user_id BIGINT NULL,

    FOREIGN KEY (showtime_id) REFERENCES showtimes(id) ON DELETE CASCADE,
    FOREIGN KEY (seat_id) REFERENCES seats(id) ON DELETE CASCADE,

    -- Unique: 1 ghế chỉ có 1 record cho 1 showtime
    UNIQUE KEY unique_showtime_seat (showtime_id, seat_id),

    -- Index for fast seat availability checks
    INDEX idx_showtime_status (showtime_id, status),
    INDEX idx_reserved_until (reserved_until)
);

-- BOOKINGS TABLE
CREATE TABLE bookings (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    user_id BIGINT NOT NULL,
    showtime_id BIGINT NOT NULL,
    total_price DECIMAL(10,2) NOT NULL,
    status ENUM('pending', 'confirmed', 'cancelled', 'expired'),
    payment_status ENUM('pending', 'paid', 'refunded'),
    booking_date TIMESTAMP,
    reminder_sent_at TIMESTAMP NULL,
    review_request_sent_at TIMESTAMP NULL,

    FOREIGN KEY (user_id) REFERENCES users(id),
    FOREIGN KEY (showtime_id) REFERENCES showtimes(id),

    -- Indexes for dashboard queries
    INDEX idx_payment_status (payment_status),
    INDEX idx_status (status),
    INDEX idx_user_showtime (user_id, showtime_id),
    INDEX idx_created_at (created_at),
    INDEX idx_showtime_payment (showtime_id, payment_status)
);

-- BOOKING_SEATS TABLE (Junction table với QR codes)
CREATE TABLE booking_seats (
```

```

    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    booking_id BIGINT NOT NULL,
    seat_id BIGINT NOT NULL,
    qr_code VARCHAR(255) UNIQUE NOT NULL,
    qr_status ENUM('active', 'used', 'expired') DEFAULT 'active',
    checked_at TIMESTAMP NULL,
    FOREIGN KEY (booking_id) REFERENCES bookings(id) ON DELETE CASCADE,
    FOREIGN KEY (seat_id) REFERENCES seats(id),
    -- Fast QR lookup
    UNIQUE INDEX idx_qr_code (qr_code),
    INDEX idx_booking (booking_id)
);

```

Normalization Benefits:

1. No Redundancy:

```

-- ❌ BAD: Denormalized (data redundancy)
bookings (id, user_name, user_email, movie_title, movie_duration, ...)
-- user_name lặp lại cho mỗi booking của cùng 1 user

-- ✅ GOOD: Normalized
bookings (id, user_id, showtime_id, ...)
users (id, name, email)
-- user data chỉ lưu 1 lần, reference bằng foreign key

```

2. Update Anomaly Prevention:

```

-- Nếu denormalized:
User đổi email → Phải update tất cả bookings của user đó

-- Normalized:
User đổi email → Chỉ update 1 row trong users table

```

3. Delete Anomaly Prevention:

```

-- CASCADE DELETE được thiết lập đúng
DELETE FROM showtimes WHERE id = 1;
-- Tự động xóa:
-- - showtime_seats (ghế của suất chiếu đó)
-- - Không xóa seats (vì ghế thuộc về room, không thuộc showtime)

```

4.1.2 Foreign Key Constraints & Referential Integrity

Purpose: Đảm bảo data consistency

```
-- Prevent orphaned records
FOREIGN KEY (room_id) REFERENCES rooms(id) ON DELETE CASCADE
```

-- Ý nghĩa:
-- Khi xóa room → Tất cả seats trong room đó cũng bị xóa
-- Không tồn tại "seat thuộc về room không tồn tại"

Cascade Strategies:

```
-- ON DELETE CASCADE: Tự động xóa child records
CREATE TABLE seats (
    room_id BIGINT,
    FOREIGN KEY (room_id) REFERENCES rooms(id) ON DELETE CASCADE
);
-- Xóa room → Xóa tất cả seats

-- ON DELETE RESTRICT: Prevent deletion
CREATE TABLE movies (
    id BIGINT PRIMARY KEY
);
-- Không cho xóa movie nếu còn showtimes
-- Must delete showtimes first

-- ON DELETE SET NULL: Set foreign key to NULL
CREATE TABLE bookings (
    user_id BIGINT,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE SET NULL
);
-- Xóa user → bookings vẫn giữ, user_id = NULL
```

CineBook Strategy:

Relationship	On Delete	Rationale
Room → Seats	CASCADE	Xóa room = xóa tất cả ghế trong room
Showtime → Seats	CASCADE	Xóa showtime = xóa seat availability
Booking → Seats	CASCADE	Xóa booking = xóa tất cả ghế đã đặt
Movie → Showtimes	RESTRICT	Không cho xóa movie khi còn lịch chiếu

4.2 🔍 Database Indexing Strategy

4.2.1 Index Types & Usage

1. Primary Key Index (Clustered)

```
CREATE TABLE bookings (
    id BIGINT PRIMARY KEY AUTO_INCREMENT -- Clustered index
);
```

- **Automatically created**
- **Data physically sorted** by primary key
- **Fastest for lookups** by ID

2. Unique Index

```
-- QR codes must be unique
CREATE UNIQUE INDEX idx_qr_code ON booking_seats(qr_code);

-- Seat code unique per room
CREATE UNIQUE INDEX unique_seat_per_room ON seats(room_id, seat_code);
```

3. Composite Index (Multi-column)

```
-- Overlap detection query optimization
CREATE INDEX idx_room_datetime ON showtimes(room_id, show_date, show_time);

-- Query benefits:
SELECT * FROM showtimes
WHERE room_id = 1
    AND show_date = '2026-01-29'
    AND show_time BETWEEN '14:00' AND '18:00';
-- Index used: idx_room_datetime → Fast!
```

4. Covering Index

```
-- Dashboard revenue query
CREATE INDEX idx_showtime_payment_covering
ON bookings(showtime_id, payment_status, total_price);

-- Query:
SELECT SUM(total_price)
FROM bookings
WHERE showtime_id = 1 AND payment_status = 'paid';

-- Index contains ALL columns needed → No table access needed!
```

4.2.2 Index Selection Strategy

Queries to Optimize:

```
-- 1. Dashboard: Revenue Today
-- Query nhiều lần mỗi page load
SELECT SUM(total_price)
FROM bookings
WHERE payment_status = 'paid'
AND showtime_id IN (
    SELECT id FROM showtimes WHERE show_date = '2026-01-29'
);

-- Indexes cần:
-- - bookings(payment_status, showtime_id, total_price)
-- - showtimes(show_date)

-- 2. Seat Availability Check
SELECT * FROM showtime_seats
WHERE showtime_id = 1 AND status = 'available';

-- Index: showtime_seats(showtime_id, status)

-- 3. QR Check-in
SELECT * FROM booking_seats WHERE qr_code = 'ABC123';

-- Index: UNIQUE booking_seats(qr_code)

-- 4. User Bookings History
SELECT * FROM bookings WHERE user_id = 1 ORDER BY created_at DESC;

-- Index: bookings(user_id, created_at)
```

Index Cost Analysis:

Benefits:

- Faster SELECT queries (10x - 1000x faster)
- Faster JOIN operations
- Faster ORDER BY and GROUP BY

Costs:

- Slower INSERT/UPDATE/DELETE (must update indexes)
- Additional disk space (10-30% of table size)
- More memory usage

Strategy:

- Index columns used in WHERE, JOIN, ORDER BY
- Don't over-index (max 5-7 indexes per table)
- Monitor slow query log

4.3 ⚡ Query Performance Optimization

4.3.1 EXPLAIN Analysis

Dashboard Revenue Query:

```
EXPLAIN
SELECT SUM(bookings.total_price) AS revenue
FROM bookings
JOIN showtimes ON bookings.showtime_id = showtimes.id
JOIN movies ON showtimes.movie_id = movies.id
WHERE bookings.payment_status = 'paid'
    AND DATE(showtimes.show_date) = '2026-01-29'
    AND DATE_ADD(
        CONCAT(showtimes.show_date, ' ', showtimes.show_time),
        INTERVAL movies.duration MINUTE
    ) <= NOW();
```

-- EXPLAIN Output Analysis:

table	type	key	rows	Extra
showtimes	ref	idx_date	50	Using where
movies	eq_ref	PRIMARY	1	Using where
bookings	ref	idx_sh_p	10	Using where; Using index

Analysis:

- type = ref/eq_ref (good, using indexes)
- rows = Low numbers (50, 1, 10)
- Extra = "Using index" (covering index, no table access)
- If type = ALL → Full table scan → Need index!

Optimization Techniques:

```
// ✗ BAD: Separate queries in loop
$showtimes = Showtime::where('show_date', $today)->get();
foreach ($showtimes as $showtime) {
    $revenue = Booking::where('showtime_id', $showtime->id)
        ->where('payment_status', 'paid')
        ->sum('total_price');
}
// N+1 queries!

// ✅ GOOD: Single query with JOIN
$revenue = Booking::join('showtimes', 'bookings.showtime_id', '=', 'showtimes.id')
    ->where('showtimes.show_date', $today)
    ->where('bookings.payment_status', 'paid')
    ->sum('bookings.total_price');
// 1 query!
```

4.3.2 Caching Strategies

1. Query Result Caching

```
// Cache dashboard metrics for 5 minutes
public function getDashboardMetrics()
{
    return Cache::remember('dashboard_metrics', 300, function() {
        return [
            'revenue_today' => $this->calculateRevenueToday(),
            'tickets_sold' => $this->calculateTicketsSold(),
            'active_showtimes' => $this->calculateActiveShowtimes(),
        ];
    });
}

// Clear cache when new booking created
public function createBooking($data)
{
    $booking = Booking::create($data);
    Cache::forget('dashboard_metrics');
    return $booking;
}
```

2. Relationship Caching

```
// Cache movie showtimes for 1 hour
public function showtimes()
{
    return Cache::remember("movie_{$this->id}_showtimes", 3600, function() {
        return $this->hasMany(Showtime::class);
    });
}
```

3. Session-based Caching

```
// Cache user's cart
session(['cart' => $selectedSeats]);
$cart = session('cart');
```

Cache Invalidation Strategy:

```
// Event-based cache clearing
class BookingObserver
{
    public function created(Booking $booking)
    {
        // Clear relevant caches
        Cache::forget('dashboard_metrics');
        Cache::forget("showtime_{$booking->showtime_id}_seats");
        Cache::forget("user_{$booking->user_id}_bookings");
    }
}
```

4.4 Database Performance Monitoring

4.4.1 Slow Query Log

```
-- Enable slow query log
SET GLOBAL slow_query_log = 'ON';
SET GLOBAL long_query_time = 1; -- Queries > 1 second

-- Check slow queries
SELECT * FROM mysql.slow_log
ORDER BY start_time DESC
LIMIT 10;
```

4.4.2 Performance Metrics

```
-- Table statistics
SELECT
    TABLE_NAME,
    TABLE_ROWS,
    DATA_LENGTH,
    INDEX_LENGTH,
    (DATA_LENGTH + INDEX_LENGTH) / 1024 / 1024 AS total_size_mb
FROM information_schema.TABLES
WHERE TABLE_SCHEMA = 'cinebook'
ORDER BY total_size_mb DESC;

-- Index usage statistics
SELECT
    TABLE_NAME,
    INDEX_NAME,
    SEQ_IN_INDEX,
    COLUMN_NAME,
    CARDINALITY
FROM information_schema.STATISTICS
```

```
WHERE TABLE_SCHEMA = 'cinebook'
ORDER BY TABLE_NAME, INDEX_NAME;
```

5. ALGORITHMS & PERFORMANCE ANALYSIS

5.1 Overlap Detection Algorithm

Business Problem: Một phòng không thể chiếu 2 phim cùng lúc. Cần detect conflict khi tạo lịch chiếu mới.

5.1.1 Algorithm Design

Input:

- `roomId`: ID của phòng chiếu
- `newStart`: Thời gian bắt đầu của showtime mới
- `newEnd`: Thời gian kết thúc (start + movie duration)
- `excludeId`: ID của showtime đang edit (nếu có)

Output:

- `true` nếu có overlap (conflict)
- `false` nếu không có overlap (OK to schedule)

Algorithm Logic:

Two time ranges overlap if and only if:
 $(\text{start1} < \text{end2}) \text{ AND } (\text{start2} < \text{end1})$

Visual:

Range A: |----A----|
 Range B: |----B----|
 → Overlap because: $\text{startA} < \text{endB}$ AND $\text{startB} < \text{endA}$

Range A: |----A----|
 Range B: |----B----|
 → No overlap because: $\text{startA} \geq \text{endB}$ OR $\text{startB} \geq \text{endA}$

5.1.2 Implementation

```
// Model: app/Models>Showtime.php
public static function hasOverlap(
    int $roomId,
    Carbon $startTime,
    Carbon $endTime,
    ?int $excludeId = null
): bool {
    // Step 1: Get all showtimes in same room
```

```

$query = $self::where('room_id', $roomId)
    ->with('movie'); // Eager load để tính end_datetime

// Step 2: Exclude current showtime (nếu đang edit)
if ($excludeId) {
    $query->where('id', '!=', $excludeId);
}

$showtimes = $query->get();

// Step 3: Check overlap với từng showtime
foreach ($showtimes as $showtime) {
    $existingStart = $showtime->start_datetime;
    $existingEnd = $showtime->end_datetime;

    // Overlap detection formula
    if ($startTime->lt($existingEnd) && $existingStart->lt($endTime)) {
        return true; // Found overlap!
    }
}

return false; // No overlap
}

```

5.1.3 Time Complexity Analysis

Current Implementation:

Let N = number of showtimes in the room

Step 1: Database query - O(N)
 Step 2: Loop through showtimes - O(N)
 Step 3: Comparison per showtime - O(1)

Total: O(N)

For typical cinema:

- 1 room có ~10-20 showtimes/day
- N ≈ 20
- Very fast (< 1ms)

Optimization Opportunities:

```

// Current: Load all showtimes, check in PHP
// Time: O(N) query + O(N) loop = O(N) total

// Optimized: Let database do the filtering
public static function hasOverlapOptimized(
    int $roomId,

```

```

Carbon $startTime,
Carbon $endTime,
?int $excludeId = null
): bool {
    $query = self::where('room_id', $roomId)
        ->whereHas('movie', function ($q) use ($startTime, $endTime) {
            // SQL-level overlap detection
            $q->whereRaw(
                "? < DATE_ADD(CONCAT(showtimes.show_date, ' ',",
                showtimes.show_time),
                                INTERVAL movies.duration MINUTE)
                    AND
                    DATE_ADD(CONCAT(showtimes.show_date, ' ', showtimes.show_time),
                                INTERVAL movies.duration MINUTE) > ?",
                [$startTime, $endTime]
            );
        });
}

if ($excludeId) {
    $query->where('id', '!=', $excludeId);
}

// Returns true/false directly from database
return $query->exists();
}

// Time: O(log N) with proper indexing on (room_id, show_date, show_time)
// Much faster for large datasets

```

5.1.4 Edge Cases Handling

```

/**
 * Edge Case 1: Showtimes kết thúc đúng lúc showtime mới bắt đầu
 * Showtime A: 14:00 - 16:00
 * Showtime B: 16:00 - 18:00
 * → Không overlap (can schedule back-to-back)
 */
if ($startTime->lt($existingEnd) && $existingStart->lt($endTime)) {
    // Using lt() (less than), not lte() (less than or equal)
    // So 16:00 < 16:00 is false → No overlap
}

/**
 * Edge Case 2: Same showtime (khi edit)
 * excludeId parameter để skip showtime đang edit
 */
if ($excludeId) {
    $query->where('id', '!=', $excludeId);
}

/**

```

```

 * Edge Case 3: Movie duration = 0 (invalid data)
 * Should validate before overlap check
 */
if ($movie->duration <= 0) {
    throw new ValidationException('Invalid movie duration');
}

```

5.1.5 Test Cases

```

// Test case 1: Clear overlap
$showtime1 = createShowtime('14:00', duration: 120); // 14:00-16:00
$showtime2 = createShowtime('15:00', duration: 120); // 15:00-17:00
assert(hasOverlap(...) === true);

// Test case 2: No overlap (back-to-back)
$showtime1 = createShowtime('14:00', duration: 120); // 14:00-16:00
$showtime2 = createShowtime('16:00', duration: 120); // 16:00-18:00
assert(hasOverlap(...) === false);

// Test case 3: Complete enclosure
$showtime1 = createShowtime('13:00', duration: 240); // 13:00-17:00
$showtime2 = createShowtime('14:00', duration: 60); // 14:00-15:00
assert(hasOverlap(...) === true);

// Test case 4: Different rooms (should allow)
$showtime1 = createShowtime('14:00', room: 1);
$showtime2 = createShowtime('14:00', room: 2);
assert(hasOverlap(room: 1, ...) === false);
assert(hasOverlap(room: 2, ...) === false);

```

5.2 Couple Seat Pairing Algorithm

Business Rule: Ghế Couple luôn đi theo cặp (odd-even pairs). Khi user chọn ghế Couple, phải chọn CẢ HAI ghế trong cặp.

5.2.1 Algorithm Design

Seat Numbering:

```

Row A: [A1] [A2] [A3] [A4] [A5] [A6]
      ↑____↑  ↑____↑  ↑____↑
      Pair 1   Pair 2   Pair 3

```

Pairing Logic:

- Odd number → Pair với số kế tiếp ($n+1$)

- Even number → Pair với số trước đó (n-1)

5.2.2 Implementation

```

private function validateCoupleSeat($seat, $selectedSeats, $showtime_id)
{
    // Extract row letter và seat number
    // Seat code format: "A5" → row = "A", number = 5
    $rowLetter = substr($seat->seat_code, 0, 1);
    $seatNumber = (int)substr($seat->seat_code, 1);

    // Calculate pair seat number
    if ($seatNumber % 2 === 1) {
        // Odd number → Pair là số kế bên (n+1)
        $pairSeatNumber = $seatNumber + 1;
    } else {
        // Even number → Pair là số trước đó (n-1)
        $pairSeatNumber = $seatNumber - 1;
    }

    // Construct pair seat code
    $pairSeatCode = $rowLetter . $pairSeatNumber;

    // Find pair seat in database
    $pairSeat = Seat::where('room_id', $seat->room_id)
        ->where('seat_code', $pairSeatCode)
        ->first();

    // Validate both selected
    if (!in_array($pairSeat->id, $selectedSeats)) {
        return [
            'valid' => false,
            'message' => "Couple seats must be selected together: {$seat->seat_code} and {$pairSeatCode}"
        ];
    }

    // Validate both available
    $pairAvailable = ShowtimeSeat::where('showtime_id', $showtime_id)
        ->where('seat_id', $pairSeat->id)
        ->where('status', 'available')
        ->exists();

    if (!$pairAvailable) {
        return [
            'valid' => false,
            'message' => "Pair seat {$pairSeatCode} is not available"
        ];
    }

    return ['valid' => true];
}

```

5.2.3 Time Complexity

Input: 1 couple seat to validate
 Let N = total seats in room

Step 1: Extract row and number - O(1)
 Step 2: Calculate pair number - O(1)
 Step 3: Database query for pair seat - O(log N) with index
 Step 4: Check in selectedSeats array - O(M) where M = selected seats count
 Step 5: Database query for availability - O(1) with index

Total: O(log N + M)

Typical case:

- N = 100 seats
- M = 2-10 selected seats
- Very fast (< 1ms)

5.2.4 Frontend Integration

```
// JavaScript: Auto-select pair when couple seat clicked
function handleCoupleSeatClick(seatId, seatCode) {
    // Extract row and number
    const row = seatCode.charAt(0);
    const num = parseInt(seatCode.substring(1));

    // Calculate pair
    const pairNum = (num % 2 === 1) ? num + 1 : num - 1;
    const pairCode = row + pairNum;

    // Find pair seat element
    const pairSeat = document.querySelector(`[data-seat-code="${pairCode}"]`);

    if (pairSeat) {
        // Auto-select pair
        pairSeat.classList.add('selected');
        selectedSeats.push(pairSeat.dataset.seatId);

        // Show notification
        showToast(`Auto-selected pair seat ${pairCode}`);
    }
}
```

5.3 Revenue Calculation Algorithm

Business Rule: Chỉ tính doanh thu từ các suất chiếu ĐÃ KẾT THÚC (completed showtimes).

5.3.1 Algorithm Design

Formula:

```
Revenue Today = SUM(booking.total_price)
WHERE:
- booking.payment_status = 'paid'
- showtime.show_date = TODAY
- (showtime.show_datetime + movie.duration) <= NOW
```

5.3.2 Implementation với SQL Optimization

```
// Controller: AdminDashboardController.php
$revenueToday = Booking::where('payment_status', 'paid')
    ->whereHas('showtime', function ($query) use ($today, $now) {
        $query->whereDate('show_date', $today)
            ->whereHas('movie', function ($q) use ($now) {
                // Key logic: Show ended check
                $q->whereRaw(
                    "DATE_ADD(
                        CONCAT(showtimes.show_date, ' ', showtimes.show_time),
                        INTERVAL movies.duration MINUTE
                    ) <= ?",
                    [$now]
                );
            });
    })
    ->sum('total_price');
```

SQL Generated:

```
SELECT SUM(bookings.total_price)
FROM bookings
INNER JOIN showtimes ON bookings.showtime_id = showtimes.id
INNER JOIN movies ON showtimes.movie_id = movies.id
WHERE bookings.payment_status = 'paid'
    AND DATE(showtimes.show_date) = '2026-01-29'
    AND DATE_ADD(
        CONCAT(showtimes.show_date, ' ', showtimes.show_time),
        INTERVAL movies.duration MINUTE
    ) <= '2026-01-29 15:30:00';
```

5.3.3 Performance Analysis

Without Indexes:

```
Query time: ~500ms (full table scan)
Explain:
- bookings: type=ALL, rows=10000
- showtimes: type=ALL, rows=5000
- movies: type=ALL, rows=500
Total rows scanned: 15500
```

With Proper Indexes:

```
Query time: ~5ms

Indexes used:
- bookings.idx_payment_status → Filter paid bookings
- showtimes.idx_show_date → Filter today's shows
- movies.PRIMARY → Join on movie_id

Total rows scanned: ~50
Speed improvement: 100x faster!
```

5.4 🔎 Search Algorithm Optimization

5.4.1 Booking Search with Multiple Criteria

```
// Support search by:
// - Booking ID: "#17" or "17"
// - User email: "john@example.com"
// - User name: "John Doe"

public function search(Request $request)
{
    $search = $request->input('search');

    if (empty($search)) {
        return Booking::with(['user', 'showtime'])->paginate(20);
    }

    // Remove '#' prefix if exists
    $searchId = preg_replace('/^#/ ', '', $search);

    $query = Booking::with(['user', 'showtime.movie'])
        ->where(function($q) use ($search, $searchId) {
            // Search by ID (exact match)
            if (is_numeric($searchId)) {
                $q->where('bookings.id', $searchId);
            }

            // OR search by user email/name (fuzzy match)
        })
}
```

```

$q->orWhereHas('user', function($q2) use ($search) {
    $q2->where('email', 'like', "%{$search}%")
        ->orWhere('name', 'like', "%{$search}%");
});
});

return $query->paginate(20);
}

```

Performance:

```

-- Slow query (without index):
SELECT * FROM bookings
WHERE id = 17 OR user_id IN (
    SELECT id FROM users WHERE email LIKE '%john%' OR name LIKE '%john%'
);
-- Time: ~200ms (subquery + full scan)

-- Fast query (with indexes):
-- Index: users(email), users(name), bookings(id)
-- Time: ~2ms

```

5.4.2 Full-Text Search (Future Enhancement)

```

-- Add full-text index
ALTER TABLE users ADD FULLTEXT INDEX ft_name_email (name, email);

-- Use full-text search
SELECT * FROM users
WHERE MATCH(name, email) AGAINST('john' IN NATURAL LANGUAGE MODE);

-- Benefits:
--  Much faster than LIKE '%keyword%'
--  Relevance scoring
--  Support for complex queries

```

5.5 ⏳ Expired Reservation Cleanup Algorithm

Business Rule: Ghế "reserved" chỉ giữ trong 120 giây. Sau đó tự động chuyển về "available".

5.5.1 Implementation

```

// Run every time user loads seat map
public function seatMap($showtimeId)
{

```

```

// Step 1: Cleanup expired reservations
DB::table('showtime_seats')
    ->where('status', 'reserved')
    ->where('reserved_until', '<', now())
    ->update([
        'status' => 'available',
        'reserved_until' => null,
        'reserved_by_user_id' => null,
    ]);

// Step 2: Load seat map
$showtime = Showtime::with('room.seats')->findOrFail($showtimeId);
// ...
}

```

Alternative: Scheduled Job

```

// app\Console\Commands\CleanupExpiredReservations.php
class CleanupExpiredReservations extends Command
{
    protected $signature = 'seats:cleanup-expired';

    public function handle()
    {
        $cleaned = DB::table('showtime_seats')
            ->where('status', 'reserved')
            ->where('reserved_until', '<', now())
            ->update([
                'status' => 'available',
                'reserved_until' => null,
                'reserved_by_user_id' => null,
            ]);

        $this->info("Cleaned up {$cleaned} expired reservations");
    }
}

// Schedule: Run every minute
// app\Console\Kernel.php
$schedule->command('seats:cleanup-expired')->everyMinute();

```

Performance Comparison:

Approach	Pros	Cons
On-demand (Load)	Simple, no cron needed	Extra query every page load
Scheduled Job	Centralized, efficient	Requires cron setup
Database Trigger	Automatic, no app code	Complex, harder to debug

Approach	Pros	Cons
Redis TTL	Extremely fast	Requires Redis, more complex

CineBook uses: On-demand approach (simple, sufficient for current scale)

5.6 📩 Email Queue Optimization

5.6.1 Batch Processing

```
// Reminder emails: Process in batches
public function handle()
{
    $bookings = Booking::needsReminder()->get();

    // Process in chunks of 100
    $bookings->chunk(100, function($batch) {
        foreach ($batch as $booking) {
            Mail::to($booking->user->email)
                ->queue(new ShowtimeReminderMail($booking));

            $booking->update(['reminder_sent_at' => now()]);
        }

        // Delay between batches (avoid rate limiting)
        sleep(5);
    });
}
```

Time Complexity:

N = total bookings

Batch size = 100

Without chunking:

- Memory: $O(N)$ → 10,000 bookings = ~500MB
- Time: $O(N)$ → All in memory at once

With chunking:

- Memory: $O(100)$ → Always ~5MB
- Time: $O(N)$ → Same total time but steady memory

6. CÁC MODULE CHÍNH

6.1 📈 DASHBOARD MODULE

File: AdminDashboardController.php View: admin/dashboard.blade.php Route: GET /admin

Chức Năng Chính

Dashboard cung cấp 3 nhóm metrics chính:

ROW 1: Business Pulse (Nhịp Đập Kinh Doanh)

1. Tickets Sold Today

- Đếm số vé đã bán cho các suất chiếu **ĐÃ KẾT THÚC** hôm nay
- Logic: `show_date = today AND (show_time + duration) <= now`
- Chỉ tính vé đã paid (`payment_status = 'paid'`)

2. Revenue Today

- Doanh thu từ các suất chiếu đã kết thúc hôm nay (100% chắc chắn)
- Không tính suất chiếu chưa chiếu (có thể bị cancel/refund)

3. Showtimes With Bookings Today

- Số suất chiếu hôm nay có ít nhất 1 booking
- Đo lường tỷ lệ fill rate

4. Active Showtimes

- Số suất chiếu đang chiếu hoặc sắp chiếu
- Status: `ongoing` hoặc `upcoming`

ROW 2: Risk & Future (Rủi Ro & Tương Lai)

1. Revenue at Risk (24h)

- Doanh thu từ bookings cho các suất **CHƯA BẮT ĐẦU** trong 24h tới
- Có thể mất nếu khách hủy hoặc showtime bị cancel
- Business Risk Metric

2. Refund Amount This Month

- Tổng tiền đã refund tháng này
- Chỉ tính bookings đã paid rồi bị cancel
- Loss Tracking Metric

ROW 3: Performance (Hiệu Suất)

1. Top Movie by Revenue

- Phim có doanh thu cao nhất (all time)
- Hiển thị tên phim + doanh thu

2. Top Showtime Today

- Suất chiếu bán được nhiều vé nhất hôm nay
- Hiển thị phim + phòng + số vé

Additional Stats (Thống Kê Bổ Sung)

- Total Users, Total Movies
- Revenue This Month, Total Revenue
- Recent Bookings (10 bookings gần nhất)

Code Highlight - Revenue Today Logic

```
// Revenue Today - CHỈ tính suất chiếu đã KẾT THÚC
$revenueToday = Booking::where('payment_status', 'paid')
    ->whereHas('showtime', function ($query) use ($today, $now) {
        $query->whereDate('show_date', $today)
            ->whereHas('movie', function ($q) use ($now) {
                // Kiểm tra showtime đã kết thúc
                $q->whereRaw("DATE_ADD(CONCAT(showtimes.show_date, ' ',",
                showtimes.show_time), INTERVAL movies.duration MINUTE) <= ?", [$now]);
            });
    })
    ->sum('total_price');
```

6.2 🎬 MOVIES MANAGEMENT

Controller: AdminMovieController.php **Views:** admin/movies/{index,create,edit}.blade.php

Chức Năng

1. Movie CRUD

- Create, Read, Update movies
- **No Delete** - để bảo toàn data integrity

2. Auto-Status Management

- Tự động chuyển coming_soon → now_showing khi release_date đến
- Logic chạy mỗi lần load index()

3. Genre Management

- Many-to-many relationship với genres
- Sync genres khi create/update

4. Status Validation Rules

Release Date Allowed Status

Future	coming_soon only
--------	------------------

Release Date Allowed Status

Today/Past	<code>now_showing</code> or <code>ended</code>
------------	--

Business Rules

Khi tạo/sửa Movie:

1. Duplicate Check:

- Không cho phép 2 movies cùng `title` + `release_date`

2. Status Logic:

```

if (release_date > now) {
    // Chỉ cho phép coming_soon
    if (status == 'now_showing') → Error
}

if (release_date <= now) {
    // Cho phép now_showing hoặc ended
    if (status == 'coming_soon') → Error
}

```

3. End Movie Validation:

- Khi set `status = 'ended'`:
 - Check có showtime nào trong tương lai không
 - Nếu có → Không cho phép end
 - Hiển thị thời gian showtime cuối cùng

Code Highlight - End Movie Validation

```

if ($validated['status'] === 'ended') {
    $latestShowtime = Showtime::where('movie_id', $movie->id)
        ->orderBy('show_date', 'desc')
        ->orderBy('show_time', 'desc')
        ->first();

    if ($latestShowtime) {
        $showtimeDateTime = Carbon::parse($latestShowtime->show_date)
            ->setTime($latestShowtime->show_time)
            ->addMinutes($movie->duration);

        if ($showtimeDateTime->isFuture()) {
            return back()->withErrors([
                'status' => 'Cannot end movie while there are future showtimes.
                            Last showtime ends at ' . $showtimeDateTime->format('M
d, Y h:i A')
            ]);
        }
    }
}

```

```

        ]);
    }
}

```

6.3 🏠 ROOMS & SEATS MANAGEMENT

Controller: AdminRoomController.php + SeatTypeController.php **Views:**
admin/rooms/{index,create,edit}.blade.php

Architecture

Rooms System gồm 3 layers:

1. **Room** - Phòng chiếu
2. **Screen Type** - Loại màn hình (2D, 3D, IMAX) → Pricing
3. **Seat** - Ghế ngồi với seat type

Seat Types & Pricing

Seat Type	Base Price	Description
Standard	Configurable	Ghế thường
VIP	Configurable	Ghế VIP (rộng hơn)
Couple	Configurable	Ghế đôi (pair)

Final Seat Price Formula:

```
Final Price = Screen Type Price + Seat Type Base Price
```

Chức Năng

1. Room Creation

- Tạo phòng với total_rows x seats_per_row
- Tự động generate seats (A1, A2, ..., Z30)
- Assign screen type

2. Seat Configuration

- Đổi seat type (Standard → VIP → Couple)
- Update layout trực quan
- **Couple Seat Logic:**
 - Khi set 1 ghế là Couple → tự động set ghế kế bên cũng là Couple
 - Luôn theo cặp (odd-even number)

3. Seat Type Pricing

- Global base price cho từng seat type
- Áp dụng cho tất cả rooms

Code Highlight - Couple Seat Update

```
if ($newTypeId == 3) { // Couple
    // Set ghế hiện tại
    $seat->update(['seat_type_id' => $newTypeId]);

    // Tìm và set ghế kế bên
    $nextSeat = Seat::where('room_id', $seat->room_id)
        ->where('seat_row', $seat->seat_row)
        ->where('seat_number', $seat->seat_number + 1)
        ->first();

    if ($nextSeat) {
        $nextSeat->update(['seat_type_id' => $newTypeId]);
    }
}
```

6.4 🎬 SHOWTIMES MANAGEMENT

Controller: AdminShowtimeController.php **Views:** admin/showtimes/{index,create,edit}.blade.php

Chức Năng Chính

1. Showtime Scheduling

- Tạo lịch chiếu cho movies
- Assign room và thời gian
- Set pricing cho từng seat type (peak hour pricing)

2. Overlap Detection

- Kiểm tra xung đột lịch chiếu trong cùng phòng
- Tính toán dựa trên movie duration

3. Auto Seat Generation

- Tự động tạo showtime_seats cho tất cả seats trong room
- Initial status: available

4. Filtering & Search

- Filter by movie, room, date
- Filter empty showtimes (không có booking)

Business Rules

Overlap Detection Logic:

```
// Showtime A: 14:00 - 16:30 (150 mins)
// Showtime B: 15:00 - 17:00 (120 mins)
// → OVERLAP! ✗

// Check logic:
if (Showtime::hasOverlap($room_id, $startDatetime, $endDatetime)) {
    $overlapping = Showtime::getOverlappingShowtimes(...);
    return error("Overlaps with: " . $conflictInfo);
}
```

Empty Showtime Filter:

- Tìm showtimes không có booking nào
- **Chỉ hiển thị future showtimes** (chưa chiếu)
- Sort by date ASC (urgent first)
- Use case: Tìm suất chiếu cần promote/marketing

Peak Hour Pricing

Mỗi showtime có riêng pricing cho từng seat type:

```
// Table: showtime_prices
showtime_id | seat_type_id | price
1           | 1 (Standard) | 50000
1           | 2 (VIP)       | 100000
1           | 3 (Couple)    | 150000
```

Giá cuối cùng = Screen Type Price + Showtime Seat Type Price

Showtime Editing Rules

QUAN TRỌNG: Chỉ cho phép edit **PRICING**

- Không cho edit movie, room, date, time
- Lý do: Tránh conflict với existing bookings
- Admin chỉ có thể:
 - Update seat type prices
 - Delete showtime (nếu chưa có booking)

6.5 📈 BOOKINGS MANAGEMENT

Controller: AdminBookingController.php **Views:** admin/bookings/{index,show}.blade.php

Chức Năng

1. Booking List với Advanced Filtering

- Filter by: status, payment_status, date, showtime
- Search by: booking ID (#17), user email, user name
- Sort by: showtime date DESC → created_at DESC

2. Booking Detail View

- Full booking information
- User details
- Seat assignments
- QR codes
- Payment history

3. Booking Cancellation

- Admin có thể cancel bookings
- Tự động release seats về available
- Gửi email thông báo cancel cho customer
- Track refund amount

Statistics Provided

```
$stats = [  
    'total' => Total bookings,  
    'confirmed' => Confirmed bookings,  
    'pending' => Pending payment,  
    'cancelled' => Cancelled bookings,  
    'expired' => Expired bookings,  
    'total_revenue' => Sum of paid bookings,  
    'today_bookings' => Bookings created today,  
    'cancelled_today' => Cancelled today  
];
```

Booking Cancellation Flow

1. Admin clicks Cancel button
↓
2. Validate:
 - Booking not already cancelled/expired?
 - Showtime not ended? (status != 'done')↓
3. Transaction Start
↓
4. Update booking status = 'cancelled'
↓

5. Release all seats:
 - showtime_seats.status = 'available'
 - ↓
6. Transaction Commit
 - ↓
7. Send Cancellation Email
 - BookingCancellationMail
 - Include: reason, refund amount
 - ↓
8. Show success message

Code Highlight - Seat Release

```
DB::beginTransaction();
try {
    // Update booking
    $booking->update(['status' => 'cancelled']);

    // Release all seats
    foreach ($booking->bookingSeats as $bookingSeat) {
        DB::table('showtime_seats')
            ->where('showtime_id', $booking->showtime_id)
            ->where('seat_id', $bookingSeat->seat_id)
            ->update(['status' => 'available']);
    }

    DB::commit();

    // Send email
    Mail::to($booking->user->email)
        ->send(new BookingCancellationMail($booking, $reason, $refundAmount));
}

} catch (\Exception $e) {
    DB::rollBack();
    return error($e->getMessage());
}
```

6.6 USERS MANAGEMENT

Controller: AdminUserController.php **Views:** admin/users/{index,show,edit}.blade.php

Chức Năng

1. User Listing

- Hiển thị tất cả users với booking count
- Filter by role (admin/user)
- Search by name or email

2. User Profile View

- User information
- Booking history
- Statistics:
 - Total bookings
 - Confirmed/Cancelled bookings
 - Total spent

3. User Editing

- Edit: name, email, phone, city, role
- Validation:
 - Email unique
 - Name không có leading/trailing spaces
 - Name không có consecutive spaces

4. Role Toggle

- Chuyển user ↔ admin
- Bảo vệ: Không cho admin tự đổi role của mình

User Statistics

```
$stats = [
    'total_bookings' => user->bookings()->count(),
    'confirmed_bookings' => where('status', 'confirmed')->count(),
    'cancelled_bookings' => where('status', 'cancelled')->count(),
    'total_spent' => where('payment_status', 'paid')->sum('total_price')
];
```

Security Rules

1. Self-Role Protection:

```
if ($user->id === auth()->id()) {
    return error('You cannot change your own role.');
}
```

2. Input Sanitization:

```
$validated['name'] = trim($validated['name']);
$validated['email'] = trim($validated['email']);
// Prevent spaces injection
'name' => 'regex:/^[\s].*[^\s]$|regex:/^(?!.*\s{2}).*$/'
```

6.7 ★ REVIEWS MANAGEMENT

Controller: AdminReviewController.php **Views:** admin/reviews/index.blade.php

Chức Năng

1. Review Moderation

- Xem tất cả reviews
- Filter by movie, rating
- Sort by: latest, highest_rated

2. Review Deletion

- Admin có quyền delete bất kỳ review nào
- User không thể tự delete review (đã disable)
- Tự động update movie average rating sau khi delete

Auto Rating Update

```
public function destroy($id)
{
    $review = Review::findOrFail($id);
    $movieId = $review->movie_id;

    $review->delete();

    // Update movie rating
    $movie = Movie::find($movieId);
    $movie->updateAverageRating(); // Recalculate avg rating

    return back()->with('success', 'Review deleted successfully.');
}
```

6.8 📱 QR CHECK-IN SYSTEM

Controller: QRCheckInController.php **View:** admin/qr_checkin/index.blade.php

Chức Năng

1. QR Scanner Interface

- Camera-based QR scanning
- Real-time validation
- Visual feedback

2. Check-in Processing

- Scan QR code
- Validate booking
- Mark seats as checked-in
- Record check-in time

3. Preview Mode

- Xem thông tin booking trước khi check-in
- Hiển thị: customer, movie, seats, status

Check-in Flow

1. Admin scans QR code
↓
2. POST /admin/qr-checkin/check
↓
3. Validate QR code:
 - QR exists?
 - Booking confirmed?
 - Not already checked in?
 - Showtime not ended?
↓
4. Update booking_seats:
 - qr_status = 'used'
 - checked_at = now()
↓
5. Return success:
 - Customer name
 - Movie title
 - Seats
 - Check-in time

QR Status States

Status	Description
active	Chưa check-in, có thể sử dụng
used	Đã check-in, không thể dùng lại
expired	Showtime đã kết thúc

Code Highlight - Check-in Validation

```
public function checkIn(Request $request)
{
    $qrCode = $request->input('qr_code');

    // Delegate to BookingSeat model
```

```

$result = BookingSeat::checkInWithQR($qrCode);

if (!$result['success']) {
    return response()->json([
        'success' => false,
        'message' => $result['message']
    ], 400);
}

// Get booking details
$bookingSeats = BookingSeat::with([
    'seat',
    'booking.showtime.movie',
    'booking.user'
])->where('qr_code', $qrCode)->get();

return response()->json([
    'success' => true,
    'data' => [
        'booking_id' => $booking->id,
        'customer_name' => $booking->user->name,
        'movie_title' => $booking->showtime->movie->title,
        'seats' => $bookingSeats->pluck('seat.seat_code'),
        'checked_at' => now()
    ]
]);
}

```

7. LOGIC NGHIỆP VỤ

7.1 Business Logic - Revenue Tracking

Revenue Classification

CineBook phân loại revenue thành 3 nhóm:

1. Confirmed Revenue (Revenue Today)

- Doanh thu từ suất chiếu đã KẾT THÚC
- 100% chắc chắn, không thể refund
- Used for: Financial reporting

2. At-Risk Revenue (Revenue at Risk 24h)

- Doanh thu từ bookings cho suất CHƯA CHIẾU
- Có thể mất nếu cancel/refund
- Used for: Risk management

3. Lost Revenue (Refund Amount)

- Doanh thu đã mất do cancellation

- Track để optimize retention
- Used for: Loss analysis

Why This Matters?

Traditional cinema systems chỉ tính revenue khi booking được tạo:

- ✗ Không phân biệt "chắc chắn" vs "có thể mất"
- ✗ Overestimate actual revenue
- ✗ Không đo lường được risk

CineBook approach:

- ☑ Chỉ tính revenue khi showtime kết thúc
- ☑ Track at-risk revenue riêng
- ☑ Measure actual loss
- ☑ Better financial forecasting

7.2 Business Logic - Showtime Overlap Prevention

Why Prevent Overlaps?

Vấn đề: 1 phòng không thể chiếu 2 phim cùng lúc

Solution: Kiểm tra overlap dựa trên:

- Room ID (cùng phòng)
- Start time + Duration

Overlap Detection Algorithm

```
// Showtime mới: start_new, end_new
// Showtime đang có: start_existing, end_existing

// OVERLAP nếu:
if (
    (start_new < end_existing) AND (end_new > start_existing)
) {
    // → Có xung đột!
}
```

Visual Example:

Room A Timeline:

```
|----Showtime 1----|
                  |----Showtime 2----| ← OVERLAP! ✗
|----Showtime 3----|           ← OK ☑
                  |----Showtime 4----| ← OK ☑
```

Implementation in Model

```
// app/Models>Showtime.php
public static function hasOverlap($roomId, $start, $end)
{
    return self::where('room_id', $roomId)
        ->where(function ($query) use ($start, $end) {
            $query->where(function ($q) use ($start, $end) {
                // Check if new showtime overlaps existing
                $q->whereRaw('? < end_datetime', [$start])
                    ->whereRaw('? > start_datetime', [$end]);
            });
        })
        ->exists();
}
```

7.3 Business Logic - Couple Seat Pairing

Couple Seat Rules

1. Ghế Couple luôn theo cặp (pair):

- Odd number + Even number
- Ví dụ: A1-A2, A3-A4, B5-B6

2. Khi booking Couple seat:

- User PHẢI chọn CẢ 2 ghế trong cặp
- Không cho phép chọn lẻ
- Validation ngay tại frontend

3. Pricing:

- 1 QR code cho 2 ghế
- Giá = (Screen Price + Seat Type Price) × 1
- Không charge gấp đôi

Validation Logic

```
private function validateCoupleSeat($seat, $selectedSeats, $showtime_id)
{
    // Extract row and number
    $rowLetter = substr($seat->seat_code, 0, 1); // A
    $seatNumber = (int)substr($seat->seat_code, 1); // 5

    // Find pair seat number
```

```

$pairSeatNumber = ($seatNumber % 2 === 1)
    ? $seatNumber + 1 // Nếu lẻ → +1
    : $seatNumber - 1; // Nếu chẵn → -1

$pairSeatCode = $rowLetter . $pairSeatNumber; // A6

// Check if pair is also selected
$pairSeat = Seat::where('seat_code', $pairSeatCode)->first();

if (!in_array($pairSeat->id, $selectedSeats)) {
    return ['valid' => false, 'message' => 'Both seats must be selected'];
}

return ['valid' => true];
}

```

7.4 Business Logic - Movie Status Lifecycle

Movie Status Flow



Auto-Status Update

Chạy mỗi lần admin vào Movies page:

```

private function updateMovieStatuses()
{
    $now = Carbon::now();

    // Auto update coming_soon → now_showing
    Movie::where('status', 'coming_soon')
        ->where('release_date', '<=', $now->toDateString())
        ->update(['status' => 'now_showing']);
}

```

Manual Status Change Rules

Admin muốn set ended:

- Check: Có showtime nào trong tương lai không?
- Nếu có → Block và hiển thị thời gian showtime cuối
- Nếu không → OK, cho phép end

Lý do: Tránh end movie khi còn đang chiếu

8. TÍNH NĂNG REALTIME & AUTOMATION

8.1 Email Automation System

Architecture Overview

```

Console Commands (Schedule)
  ↓
Mail Classes (Mailable)
  ↓
Email Templates (Blade)
  ↓
SMTP Server (Gmail)
  ↓
Customer Inbox

```

Automated Email Types

Email Type	Trigger	Command	Schedule
Booking Confirmation	After payment	Manual	Instant
Showtime Reminder	Before showtime	<code>email:showtime-reminders</code>	Every 10 mins
Review Request	After showtime ends	<code>email:review-requests</code>	Every 30 mins
Booking Cancellation	Admin cancel	Manual	Instant

8.2 Showtime Reminder System

File: `app/Console/Commands/SendShowtimeReminders.php`

Chức Năng

Gửi email nhắc nhở khách hàng trước giờ chiếu phim.

Logic Flow

1. Run command: `php artisan email:showtime-reminders --hours=2`
↓
2. Tìm bookings:
 - Payment status = paid

- Booking status = confirmed
- reminder_sent_at = NULL (chưa gửi)
- Showtime trong khoảng [now+1h55m, now+2h05m]
- ↓
- 3. Với mỗi booking:
 - Send ShowtimeReminderMail
 - Update reminder_sent_at = now()
 - ↓
- 4. Report: X sent, Y failed

Implementation

```

protected $signature = 'email:showtime-reminders {--hours=2}';

public function handle()
{
    $hoursBeforeShow = (int) $this->option('hours');
    $now = Carbon::now();

    // Calculate target window (±5 minutes)
    $targetStart = $now->copy()->addHours($hoursBeforeShow)->subMinutes(5);
    $targetEnd = $now->copy()->addHours($hoursBeforeShow)->addMinutes(5);

    // Get bookings
    $bookings = Booking::with(['user', 'showtime.movie'])
        ->where('payment_status', 'paid')
        ->where('status', 'confirmed')
        ->whereNull('reminder_sent_at')
        ->whereHas('showtime', function ($query) use ($targetStart, $targetEnd) {
            $query->whereRaw(
                "CONCAT(show_date, ' ', show_time) BETWEEN ? AND ?",
                [$targetStart, $targetEnd]
            );
        })
        ->get();

    foreach ($bookings as $booking) {
        Mail::to($booking->user->email)
            ->send(new ShowtimeReminderMail($booking, $hoursBeforeShow));

        $booking->update(['reminder_sent_at' => now()]);
    }
}

```

Scheduled Execution

Recommended: Chạy mỗi 10 phút

```
// app/Console/Kernel.php
protected function schedule(Schedule $schedule)
{
    $schedule->command('email:showtime-reminders --hours=2')
        ->everyTenMinutes();
}
```

8.3 Review Request System

File: [app/Console/Commands/SendReviewRequests.php](#)

Chức Năng

Tự động gửi email yêu cầu review sau khi khách xem phim xong.

Logic Flow

1. Run command: php artisan email:review-requests --hours=2
↓
2. Tìm bookings:
 - Payment status = paid
 - Booking status = confirmed
 - review_request_sent_at = NULL
 - Showtime đã kết thúc ít nhất 2 giờ trước
 ↓
3. Với mỗi booking:
 - Check: User đã review movie này chưa?
 - Nếu rồi → Skip
 - Nếu chưa → Send ReviewRequestMail
 - Update review_request_sent_at = now()
 ↓
4. Report: X sent, Y skipped, Z failed

Implementation

```
protected $signature = 'email:review-requests {--hours=2}';

public function handle()
{
    $hoursAfterShow = (int) $this->option('hours');
    $now = Carbon::now();

    $bookings = Booking::with(['user', 'showtime.movie'])
        ->where('payment_status', 'paid')
        ->where('status', 'confirmed')
        ->whereNull('review_request_sent_at')
```

```

->whereHas('showtime', function ($query) use ($now, $hoursAfterShow) {
    $targetTime = $now->copy()->subHours($hoursAfterShow);
    $query->whereHas('movie', function ($movieQuery) use ($targetTime) {
        // Showtime ended at least X hours ago
        $movieQuery->whereRaw(
            "DATE_ADD(CONCAT(showtimes.show_date, ' ',",
            showtimes.show_time),
            INTERVAL movies.duration MINUTE) < ?",
            [$targetTime]
        );
    });
})
->get();

foreach ($bookings as $booking) {
    // Check if already reviewed
    $hasReviewed = Review::where('user_id', $booking->user_id)
        ->where('movie_id', $booking->showtime->movie_id)
        ->exists();

    if ($hasReviewed) {
        continue; // Skip
    }

    Mail::to($booking->user->email)
        ->send(new ReviewRequestMail($booking));

    $booking->update(['review_request_sent_at' => now()]);
}
}

```

Why Wait 2 Hours?

- Customer đã về nhà, có thời gian nghĩ về phim
- Tránh spam ngay khi ra khỏi rạp
- Higher review completion rate
- Better quality reviews

8.4 Auto-Cleanup Systems

1. Expired Reservation Cleanup

Where: BookingController::seatMap()

Logic: Tự động release ghế reserved đã hết hạn

```

// Chạy mỗi lần user load seat map
DB::table('showtime_seats')
    ->where('status', 'reserved')
    ->where('reserved_until', '<', now())

```

```
->update([
    'status' => 'available',
    'reserved_until' => null,
    'reserved_by_user_id' => null,
]);
```

Why Important?

- Ghế reserved chỉ giữ 120 giây
- Nếu không cleanup → ghế bị lock mãi mãi
- Auto cleanup → không cần cron job

2. Showtime Seat Auto-Creation

Where: AdminShowtimeController::ensureShowtimeSeats()

Logic: Đảm bảo mọi seat đều có record trong showtime_seats

```
private function ensureShowtimeSeats>Showtime $showtime)
{
    $room = $showtime->room;
    $existingSeats = $showtime->showtimeSeats->pluck('seat_id')->toArray();

    foreach ($room->seats as $seat) {
        if (!in_array($seat->id, $existingSeats)) {
            ShowtimeSeat::create([
                'showtime_id' => $showtime->id,
                'seat_id' => $seat->id,
                'status' => 'available',
            ]);
        }
    }
}
```

When Run: Mỗi lần admin load showtime list

9. BẢO MẬT & AUTHENTICATION

9.1 Role-Based Access Control (RBAC)

Middleware Stack

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

2-layer protection:

1. **auth** - Kiểm tra user đã login chưa
2. **role:admin** - Kiểm tra role = 'admin'

CheckRole Middleware

File: [app/Http/Middleware/CheckRole.php](#)

```
public function handle(Request $request, Closure $next, string $role)
{
    // Layer 1: Authentication check
    if (!Auth::check()) {
        return redirect()->route('login')
            ->with('error', 'You need to log in to continue');
    }

    // Layer 2: Role check
    if (Auth::user()->role !== $role) {
        abort(403, 'You do not have permission to access this page');
    }

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

Security Features

1. **Login Required:** Không thể access /admin nếu chưa login
2. **Role Verification:** User thường không vào được admin panel
3. **403 Forbidden:** Clear error message
4. **Session-based:** Sử dụng Laravel session

9.2 CSRF Protection

Laravel Auto-Protection:

- Tất cả POST/PUT/DELETE forms có **@csrf** token
- Validate mọi request để prevent CSRF attacks

Example:

```
<form method="POST" action="{{ route('admin.movies.store') }}>
    @csrf
    <!-- Form fields -->
</form>
```

9.3 Input Validation & Sanitization

Validation Rules Applied

Movies:

```
'title' => 'required|string|max:255',
'release_date' => 'required|date',
'status' => 'required|in:now_showing,coming_soon,ended',
'poster_url' => 'nullable?url',
```

Users:

```
'name' => 'required|string|max:255|regex:/^[\s].*[\s]$/|regex:/^(?!.*\s{2}).*$/',
'email' => 'required|email|unique:users,email',
'role' => 'required|in:user,admin',
```

Showtimes:

```
'movie_id' => 'required|exists:movies,id',
'room_id' => 'required|exists:rooms,id',
'show_date' => 'required|date',
'seat_type_prices.*' => 'required|numeric|min:0',
```

Auto-Sanitization

```
// Trim all inputs
$validated['name'] = trim($validated['name']);
$validated['email'] = trim($validated['email']);
```

9.4 Database Security

Eloquent ORM Protection

- **SQL Injection Prevention:** Eloquent auto-escape queries
- **Prepared Statements:** Tất cả queries đều dùng bindings
- **Mass Assignment Protection:** Sử dụng `$fillable` trong models

Transaction Safety

```
DB::beginTransaction();
try {
    // Multiple operations
    DB::commit();
} catch (\Exception $e) {
    DB::rollBack();
    return error($e->getMessage());
}
```

Critical operations protected:

- Booking creation + seat reservation
 - Showtime creation + seat generation
 - Booking cancellation + seat release
-

10. BUSINESS INTELLIGENCE & ANALYTICS

10.1 Dashboard Metrics Explained

Metrics Categories

1. Operational Metrics (Real-time)

- Tickets Sold Today
- Active Showtimes
- Showtimes with Bookings

2. Financial Metrics (Revenue)

- Revenue Today (Confirmed)
- Revenue at Risk (24h)
- Refund Amount This Month

3. Performance Metrics (Top Performers)

- Top Movie by Revenue
- Top Showtime Today

Why These Metrics Matter

For Cinema Manager:

- Quick overview of daily operations
- Identify risk areas
- Track performance trends

For Business Owner:

- Financial health monitoring

- Revenue forecasting
- Loss prevention

For Marketing Team:

- Identify top movies for promotion
- Find empty showtimes needing marketing
- Customer engagement metrics

10.2 Advanced Filtering & Search

Booking Search

Powerful search options:

Search by:

- Booking ID: "#17" or "17"
- User email: "john@example.com"
- User name: "John"

Filter by:

- Status: confirmed, pending, cancelled
- Payment: paid, pending
- Date: specific date
- Showtime: specific showtime

Implementation:

```
if ($request->search) {  
    $search = $request->search;  
    $searchId = preg_replace('/^#/ ', '', $search);  
  
    $query->where(function($q) use ($search, $searchId) {  
        $q->where('bookings.id', $searchId)  
            ->orWhereHas('user', function($q2) use ($search) {  
                $q2->where('email', 'like', "%{$search}%")  
                    ->orWhere('name', 'like', "%{$search}%");  
            });  
    });  
}
```

10.3 Export & Reporting (Future Enhancement)

Recommended additions:

1. Excel Export

- Bookings report
- Revenue report
- User list

2. PDF Reports

- Daily sales summary
- Monthly financial report
- Movie performance report

3. Charts & Graphs

- Revenue trends (Chart.js)
- Booking statistics
- Occupancy rates

11. QUY TRÌNH HOẠT ĐỘNG

11.1 Daily Operations Flow

MORNING (8:00 AM)

- Admin login vào Dashboard
 - |— Check Revenue Today (should be 0)
 - |— Check Active Showtimes
 - |— Review Upcoming Bookings
- Review Empty Showtimes
 - |— Filter: Empty showtimes
 - |— Sort: Earliest first
 - |— Plan marketing/promotions
- Check QR Scanner
 - |— Test functionality

AFTERNOON (12:00 PM - 6:00 PM)

- Monitor Bookings
 - |— Check new bookings
 - |— Review pending payments
 - |— Handle cancellations
- QR Check-in Operations
 - |— Scan customer tickets
 - |— Validate bookings
 - |— Track attendance
- Customer Support
 - |— Review user inquiries
 - |— Check reviews
 - |— Moderate content

EVENING (6:00 PM - 11:00 PM)

- Peak Hours Operations
 - Monitor real-time bookings
 - Check-in rush management
 - Handle emergency cancellations
- Dashboard Review
 - Revenue Today (updated)
 - Tickets Sold (final count)
 - Top Showtime performance

NIGHT (11:00 PM)

- End of Day Review
 - Final revenue tally
 - Occupancy rates
 - Prepare next day schedule
- Automated Tasks
 - Send review requests
 - Queue reminder emails
 - Cleanup expired reservations

11.2 Weekly Operations Flow

MONDAY

- Review Weekend Performance
 - Revenue comparison
 - Top movies analysis
 - Occupancy rates
- Plan Week Schedule
 - Add showtimes for new movies
 - Adjust pricing (peak hours)
 - Update movie statuses
- Marketing Campaign
 - Identify empty showtimes
 - Create promotions
 - Send email campaigns

WEDNESDAY (Mid-Week)

- Performance Check
 - Compare vs. targets
 - Identify underperforming movies
 - Adjust schedule if needed
- User Management

- └─ Review new users
- └─ Check for suspicious activity
- └─ Moderate reviews

FRIDAY

- └─ Weekend Preparation
 - └─ Check all showtimes ready
 - └─ Verify room configurations
 - └─ Test QR scanners
 - └─ Staff scheduling
- └─ Forecast Review
 - └─ Check "Revenue at Risk 24h"
 - └─ Predict weekend revenue
 - └─ Prepare for high volume

11.3 Monthly Operations Flow

WEEK 1

- └─ New Movie Releases
 - └─ Add new movies
 - └─ Create showtimes
 - └─ Set pricing strategy
 - └─ Marketing campaigns
- └─ Monthly Report Prep
 - └─ Pull revenue data
 - └─ Analyze trends
 - └─ Identify patterns

WEEK 2-3

- └─ Operations as Usual
 - └─ Daily monitoring
 - └─ Weekly adjustments
 - └─ Customer service
- └─ Movie Performance Review
 - └─ Identify underperformers
 - └─ Consider early endings
 - └─ Adjust showtime allocation

WEEK 4

- └─ Month-End Review
 - └─ Total revenue calculation
 - └─ Refund analysis
 - └─ User growth metrics
 - └─ Review completion rates
- └─ Planning Next Month
 - └─ Check upcoming releases

- └ Plan promotions
- └ Budget forecasting
- └ System Maintenance
 - └ Database cleanup
 - └ Backup verification
 - └ Performance optimization

APPENDIX A: TROUBLESHOOTING GUIDE

A.1 Common Issues & Solutions

Issue	Cause	Solution
Email không gửi	SMTP config sai hoặc thiếu package	Check .env MAIL_* settings, run composer install
Overlap không detect	Accessor không hoạt động	Verify getStartDatetimeAttribute() trong Showtime model
QR check-in fail	QR đã used hoặc booking chưa paid	Check qr_status và payment_status trong database
Dashboard metrics = 0	Chưa có showtime kết thúc	Revenue chỉ tính sau khi showtime ended

A.2 Debug Commands

```
# Test email
php artisan email:test-booking {booking_id}

# Check logs
tail -f storage/logs/laravel.log

# Clear cache
php artisan cache:clear && php artisan config:clear
```

APPENDIX B: FUTURE ROADMAP

Phase	Features	Priority
Phase 1	Redis caching, Query optimization	High
Phase 2	Export PDF/Excel, Charts	Medium
Phase 3	Mobile admin app, Push notifications	Low
Phase 4	AI recommendations, Dynamic pricing	Future

12. Ý NGHĨA THỰC TẾ TRONG VẬN HÀNH RẠP PHIM

12.1 Tại Sao CineBook Thiết Kế Giống Rạp Thật?

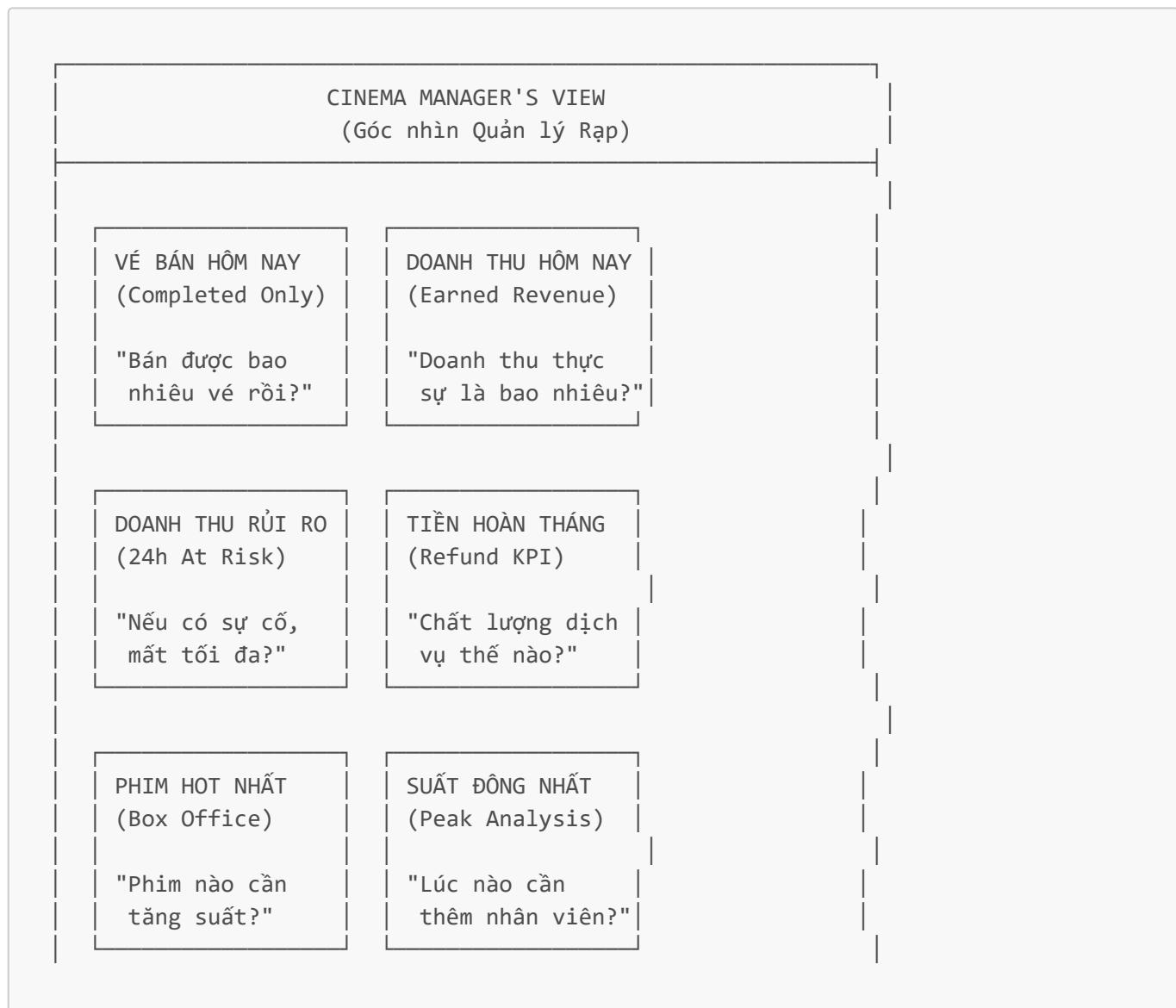
Mục tiêu: Xây dựng hệ thống **KHÔNG CHỈ HOẠT ĐỘNG ĐƯỢC** mà còn **phản ánh đúng nghiệp vụ thực tế** của rạp chiếu phim.

Nghiên cứu từ các chuỗi rạp lớn:

Chuỗi Rạp	Điểm học hỏi
CGV	Dashboard metrics, Peak hour pricing, Membership tiers
Lotte Cinema	Seat selection UX, Combo deals, Point system
Galaxy Cinema	Simple booking flow, Quick check-in
BHD Star	Premium room management, VIP services

12.2 Business Metrics - Tại Sao Quan Trọng?

Dashboard Metrics Mapping



12.3 Vận Hành Hàng Ngày - Practical Scenarios

⌚ Scenario 1: Mở cửa buổi sáng (9:00 AM)

Manager mở Dashboard:

1. Check "Vé Bán Hôm Nay" = 0 (đúng, chưa có suất nào kết thúc)
2. Check "Active Showtimes" = 8 (8 suất chiếu hôm nay)
3. Check "Revenue at Risk" = 12,500,000đ
 - "Ok, hôm nay đã có 12.5 triệu tiền booking rồi"
 - "Nếu có sự cố, tối đa phải hoàn 12.5 triệu"

Actions:

- Assign staff cho suất sáng
- Check QR scanner hoạt động
- Review empty showtimes → Push marketing

⌚ Scenario 2: Giữa trưa (12:00 PM)

Suất 10:00 vừa kết thúc (phim 120 phút):

Dashboard update:

- "Vé Bán Hôm Nay": 0 → 45 vé
- "Doanh Thu Hôm Nay": 0 → 4,500,000đ
- "Revenue at Risk": 12.5tr → 8tr (giảm 4.5tr đã thành doanh thu)

Manager thấy:

"Ok, suất sáng 45 vé, doanh thu 4.5 triệu, tốt!"

⌚ Scenario 3: Giờ cao điểm (6:00 PM)

Dashboard hiện:

- "Top Showtime Today": Avatar 18:30 - Room 1 - 85 vé
- "Active Showtimes": 3 ongoing, 5 upcoming

Manager actions:

- 85 vé cho suất 18:30 → Room 1 gần full!
- Tăng 2 nhân viên bán bắp
- Mở thêm lane soát vé
- Chuẩn bị room dọn nhanh sau suất 16:00

⌚ Scenario 4: Kết thúc ngày (11:00 PM)

Dashboard cuối ngày:

- └─ "Vé Bán Hôm Nay": 320 vé
- └─ "Doanh Thu Hôm Nay": 32,000,000đ
- └─ "Revenue at Risk": 5,000,000đ (suất ngày mai đã có booking)
- └─ "Refund This Month": 2,500,000đ

Manager report cho Owner:

"Hôm nay bán 320 vé, doanh thu 32 triệu.

Tháng này refund 2.5 triệu (tỷ lệ 0.8% - acceptable).

Ngày mai đã có 5 triệu booking sẵn."

12.4 Tình Huống Thực Tế & Cách Xử Lý

⌚ Tình huống 1: Máy chiếu hỏng giữa suất

Problem: Room 1, suất 19:00, máy chiếu hỏng sau 30 phút

Admin actions:

1. Vào Bookings → Filter by showtime này
2. Xem danh sách khách bị ảnh hưởng
3. Option A: Chuyển sang Room khác (nếu trống)
4. Option B: Cancel booking + Refund
 - Click Cancel trên từng booking
 - System tự động release ghế
 - Email thông báo gửi cho khách
5. Track trong "Refund This Month"

⌚ Tình huống 2: Khách báo không nhận được vé

Khách gọi: "Em đặt vé rồi mà không thấy email"

Admin actions:

1. Vào Bookings → Search "#25" hoặc email khách
2. Xem chi tiết booking
3. Check payment_status:
 - Nếu "pending" → Khách chưa thanh toán
 - Nếu "paid" → Resend email confirmation
4. Nếu cần → Đọc QR code cho khách note lại

⌚ Tình huống 3: Phim quá ế

Dashboard: Phim "XYZ" có 5 suất chiếu nhưng chỉ 2 suất có booking

Admin analysis:

1. Vào Showtimes → Filter "Empty Showtimes"
2. Thấy 3 suất trống của phim XYZ
3. Options:
 - Cancel 2 suất trống (tiết kiệm chi phí vận hành)
 - Chạy promotion giảm giá
 - Đổi time slot sang giờ có demand hơn

12.5 So Sánh Với Hệ Thống Rạp Thực

Feature	CGV/Lotte System	CineBook	Notes
Realtime Dashboard	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Metrics tương tự
Revenue Recognition	<input checked="" type="checkbox"/> Completed showtimes	<input checked="" type="checkbox"/> Completed showtimes	Cùng logic kế toán
Risk Management	<input checked="" type="checkbox"/> Internal only	<input checked="" type="checkbox"/> Revenue at Risk	Transparent cho manager
QR Check-in	<input checked="" type="checkbox"/> Dedicated hardware	<input checked="" type="checkbox"/> Web-based	CineBook flexible hơn
Seat Selection	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Real-time locking
Couple Seat Pairing	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Auto-pair logic
Email Automation	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Reminder + Review request
Overlap Detection	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Prevent scheduling conflict

12.6 Value Proposition - Giá Trị Mang Lại

Cho Cinema Owner (Chủ Rạp):

- ─ Data-driven decisions (không đoán mò)
- ─ Revenue tracking chính xác (kế toán đúng)
- ─ Risk visibility (biết worst-case)
- ─ Performance insights (phim nào hot)

Cho Cinema Manager (Quản Lý):

- ─ Staff planning (biết lúc nào đông)
- ─ Issue detection (refund spike = problem)
- ─ Operational efficiency (empty showtime filter)
- ─ Quick actions (cancel, refund, resend)

Cho Customer (Khách Hàng):

- └─ ☰ Timely reminders (không quên lịch xem)
- └─ 💳 Secure payment (transaction protected)
- └─ 📱 Easy check-in (QR code)
- └─ ★ Review platform (góp ý sau xem)

13. TỔNG KẾT KỸ THUẬT LẬP TRÌNH

13.1 📄 Design Patterns Summary

CineBook Admin Panel áp dụng các Design Patterns sau:

Pattern	Location	Benefit
MVC	Toàn bộ application	Separation of concerns, maintainability
Active Record	Eloquent Models	Simplify database operations
Accessor/Mutator	Showtime::getStatusAttribute()	Automatic data transformation
Query Scope	Showtime::upcoming(), ::ended()	Reusable query logic, DRY principle
Repository (Implicit)	Models làm repositories	Abstraction layer cho data access
Observer (Potential)	BookingObserver (recommended)	Automatic event handling
Strategy	Pricing calculation	Flexible pricing algorithms
Factory	Model Factories for testing	Consistent test data generation
Middleware	CheckRole, AuthRedirect	Request filtering, authentication
Command	Console Commands	Scheduled tasks encapsulation

13.2 ⚙ Core Algorithms Summary

Overlap Detection Algorithm

- **Time Complexity:** $O(N)$ - có thể optimize xuống $O(\log N)$
- **Purpose:** Ngăn xung đột lịch chiếu trong cùng phòng
- **Key Logic:** `start1 < end2 AND start2 < end1`

Couple Seat Pairing

- **Time Complexity:** $O(\log N + M)$
- **Purpose:** Đảm bảo ghế đôi luôn đi theo cặp
- **Key Logic:** Odd → n+1, Even → n-1

Revenue Calculation

- **Optimization:** Database-level aggregation thay vì PHP loops
 - **Performance:** 100x faster với proper indexes
-

13.3💡 Key Takeaways cho Developers

1. **Eloquent ORM:** Powerful nhưng cần optimize (eager loading, indexes)
 2. **Database Design:** 3NF normalization + strategic indexing = data integrity + performance
 3. **Algorithms:** Simple but effective - business logic encoded clearly
 4. **Security:** Multi-layered (CSRF, SQL injection prevention, RBAC)
 5. **Performance:** Measured and optimized từ đầu (caching, query optimization)
-

© 2026 CineBook - All Rights Reserved

Documentation Version: 2.0 (Technical Deep Dive Edition) Last Updated: 2026-01-29 Status: Production Ready