PHP/Laravel System: Secure JWT Auth, Nested Categories, and Concurrency-Safe Ticketing

Overview

This project implements a secure, high-performance API for a ticketing system. It includes JWT authentication with refresh tokens, a nested category structure using the Materialized Path pattern, concurrency-safe ticket purchases, and a React + shadcn frontend.

Scenario

Users can browse and buy tickets categorized under a nested structure (e.g., Sports > Football > Premier League). Authentication uses JWT with refresh support, and race conditions are prevented during purchases using DB locks. The system supports high concurrency and large datasets.

Design Patterns Used

1. Service Pattern – Used for `JwtService`

Justification:
The **Service pattern** was applied to encapsulate all JWT token logic (signing, verification, encoding, decoding) within a dedicated class: `JwtService`. This helps keep the controller and actions free from cryptographic logic or token structure concerns, making the codebase more **modular and reusable**. *Used in:* `App\Services\JwtService`

2. Facade Pattern – Applied to expose `JwtService` globally as `Jwt::`

Justification:
To make the `JwtService` conveniently accessible throughout the app (like in Actions), it is exposed using a **Facade**. This hides the instantiation and binding logic, providing a clean and expressive static interface (`Jwt::sign()` , `Jwt::verify()`). *Used as:* `App\Facades\Jwt`

3. Action Pattern – Used in `AttemptLoginAction`

Justification:
The **Action pattern** was used to isolate the **login logic with rate limiting and brute-force protection** in a dedicated invokable class. This aligns with the **Single Responsibility Principle**. *Used in:* `App\Actions\AttemptLoginAction`

Performance Optimization

1. Chunked CSV Import

Streamed and inserted 50k+ category records using chunked batching.

Benchmark
Command: `php artisan app:import-categories-from-csv storage/app/data/categories.csv`
Time: 1.91s | Memory: 0.01MB | SQL Queries: 103 | Rows: 51,001

2. Materialized Path for Category Trees

Supports fast subtree and breadcrumb queries using indexed path values.

3. Pagination

Improves API responsiveness for large record sets.

4. Caching for Login Throttling

Used Laravel Cache to apply exponential backoff on failed logins.

5. Database Indexes

Optimized lookups on `email`, `user_id`, `path`, and `parent_id`.

Security Measures Implemented

1. SQL Injection Protection

Used Eloquent and parameter binding throughout.

2. Mass Assignment Protection

Models use `$fillable`; `$guarded = []` avoided.

3. Rate Limiting & Brute Force Protection

Login attempts limited and delayed using cache and exponential backoff.

4. Password Policy

Used a context-aware rule to validate passwords against user info, dictionary words, and behavior context (public IP, night hours).

5. CSRF Protection

Access tokens stored in memory; refresh tokens in `HttpOnly` cookies with `SameSite=Lax` and `Secure` (in production).

6. Race Condition Mitigation

Used `lockForUpdate()` in transactions to prevent double booking. Verified using:

```
hey -n 100 -c 100 -m POST http://localhost:8000/api/tickets/3/buy
```

and Apache JMeter for high-load simulation.

Other Key Features

- RESTful API CRUD for nested categories with Materialized Path
- Bulk CSV import with memory optimization
- International phone number validation
- Soft delete with data preservation
- Api for secure PDF-only file upload
- GDPR-compliant audit logging