Using Laravel Telescope to Debug Performance Issues

Using Laravel Telescope to Debug Performance Issues

When your Laravel app slows down under traffic, it can be difficult to know whether the issue is database queries, cache misses, slow jobs, or external APIs. **Laravel Telescope** is a powerful debugging assistant that gives you full visibility into what your app is doing in real time. In this guide, we'll install Telescope, explore its dashboard, and learn how to use it to identify and fix performance bottlenecks.

1 – Install Telescope

Telescope is a Laravel package maintained by the core team. It's usually installed as a development dependency, but can also be deployed in production with proper authentication.

```
composer require laravel/telescope --dev
php artisan telescope:install
php artisan migrateCode language: Bash (bash)
```

This installs Telescope, publishes its assets, and sets up the necessary database tables to store logs and metrics.

2 – Protecting the Telescope Dashboard

The dashboard is available at `/telescope`. You should restrict access to only admins or local environments for security.

```
// app/Providers/TelescopeServiceProvider.php
protected function gate()
{
    Gate::define('viewTelescope', function ($user) {
        return in_array($user->email, [
            'admin@example.com',
        ]);
    });
}Code language: PHP (php)
```

This gate ensures only whitelisted users can access Telescope in production. In local environments, it's usually open by default.

3 – Monitoring Queries

Telescope shows every query executed in your app, with bindings and execution times. This helps spot **N+1 problems** and missing indexes quickly.

```
// Example of an N+1 problem
$users = User::all();

foreach ($users as $user) {
    echo $user->posts->count(); // triggers extra query for each user
}Code language: PHP (php)
```

Telescope will display hundreds of queries in this case, showing the N+1 issue. The fix is eager loading:

```
// Optimized with eager loading
$users = User::with('posts')->get();

foreach ($users as $user) {
    echo $user->posts->count(); // no extra queries
}Code language: PHP (php)
```

Now Telescope will show just two queries: one for users, one for posts. For more on this technique, see Eager Loading vs Lazy Loading in Laravel: Best Practices.

4 – Tracking Requests & Exceptions

Telescope records every incoming request and any exceptions thrown. This is extremely useful for debugging performance-related errors.

```
// Example: log request info in Telescope
Route::get('/profile', function () {
    return view('profile');
});Code language: PHP (php)
```

When visiting `/profile`, Telescope shows request duration, middleware stack, and queries executed. Exceptions appear in a separate tab, making it easy to trace issues.

5 – Monitoring Queues & Jobs

Telescope integrates with Laravel's queue system. It shows processed jobs, pending jobs, and failures in real time. This complements queues and Horizon, giving you both deep

debugging and production monitoring tools.

```
// Dispatch a queued job
SendWelcomeEmail::dispatch($user);Code language: PHP (php)
```

After dispatching a job, check Telescope's **Jobs** tab. You'll see job payload, execution time, and retry history—perfect for debugging job performance.

6 – Insights into Cache & Redis

Telescope logs cache hits and misses. This is critical when evaluating caching strategies for high-traffic apps.

```
// Example: cached query
Cache::remember('users.active', 60, function () {
    return User::active()->get();
});Code language: PHP (php)
```

Telescope shows whether the result came from cache or DB. This makes it easier to validate that caching is working and saving resources.

7 – Profiling Performance

Telescope records timeline data for every request: middleware duration, query time, job dispatch delays, etc. This is invaluable for pinpointing slow points in your app's lifecycle.

Combine Telescope insights with Octane to spot memory leaks and performance issues in

persistent worker setups.

Wrapping Up

Laravel Telescope provides real-time visibility into queries, requests, jobs, cache, and exceptions. It's a must-have tool for debugging performance issues in development and can be secured for production use. With Telescope, you can spot bottlenecks like N+1 queries, missing indexes, or slow jobs before they impact your users.

What's Next

- 10 Proven Ways to Optimize Laravel for High Traffic — overview of caching, queues, and scaling.
- How to Use Laravel Queues for Faster Performance — improve response times by moving tasks into queues.
- Caching Strategies in Laravel: Redis vs Database vs File — ensure your cache strategy supports high performance.