Debugging Laravel Applications with Ray and Telescope

Debugging is a critical part of building reliable Laravel applications. While `dd()` and `dump()` are quick solutions, they don't scale well for complex apps. Laravel offers powerful debugging tools like **Telescope** for application-wide monitoring, and third-party tools like **Ray** to simplify debugging during development. In this article, we'll explore both tools, show how to set them up, and walk through real-world debugging workflows.

Installing Ray for Debugging

```bash
composer require spatie/laravel-ray --devCode language: Bash (bash)
```

Ray is a desktop app by Spatie that displays debugging output in a clear, interactive interface. Once installed, you can use the `ray()` helper function anywhere in your Laravel app. Ray captures variables, queries, jobs, and more — without polluting your HTML responses or logs.

Basic Usage

```php
ray('Hello from Laravel');
ray($user);
ray($request->all());Code language: PHP (php)
```

This sends text, objects, or arrays to the Ray desktop client. It works like `dd()` but doesn't halt execution. You can also chain styles and colors to organize debugging output.

Debugging Queries with Ray

```php
// In AppServiceProvider or a debug-only service provider
\Illuminate\Support\Facades\DB::listen(function ($query) {
    ray($query->sql, $query->bindings, $query->time);
});Code language: PHP (php)
```

Ray displays executed queries, bindings, and execution time. This makes query optimization easier during development.

Debugging Jobs and Events

```
ray()->showJobs();
ray()->showEvents();Code language: PHP (php)
```

Ray can automatically show when jobs are dispatched or events are fired, helping you track async flows in your app.

Installing and Using Laravel Telescope

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

Telescope is Laravel’s official debugging assistant. It provides a dashboard at `/telescope` where you can view requests, jobs, events, queries, logs, cache hits, and more. It’s particularly powerful for tracking production issues when paired with access restrictions.

Tracking Requests and Responses

```
// Example: telescope shows request lifecycle
GET /posts/1
- Controller: PostController@show
- Queries: 2 (45ms)
- Response: 200 OKCode language: PHP (php)
```

The Requests tab in Telescope shows complete details about each incoming request, including middleware, response time, and database queries executed during that request.

Monitoring Queries

Telescope logs all queries executed in your app, including N+1 query problems. It highlights slow queries, so you can quickly identify bottlenecks.

Jobs, Events, and Cache

From the dashboard, you can see when jobs are dispatched, events are triggered, and cache hits or misses occur. This visibility helps debug background tasks and performance issues.

Securing Telescope in Production

Telescope can expose sensitive information. Protect it with authorization gates:

```php
// app/Providers/TelescopeServiceProvider.php
use Laravel\Telescope\Telescope;

Telescope::auth(function ($request) {
    return in_array($request->user()?->email, [
        'admin@example.com',
    ]);
});
```
Code language: PHP (php)

This ensures that only authorized developers can access Telescope in production environments. Always secure it before deployment.

Ray vs Telescope: When to Use Each

Feature	Ray	Telescope
Scope	Developer-focused debugging	App-wide request monitoring
Best for	Quick dumps, inspecting variables, dev feedback	Tracking queries, jobs, events, logs
UI	Desktop client	Web dashboard at /telescope
Environment	Local development only	Local + Production (with access control)
Integration	Simple ray() calls in code	Automatic hooks into Laravel internals

Use Ray for quick, developer-centric debugging. Use Telescope for request and system-wide insights. Many teams run both in parallel: Ray locally, Telescope in staging/production.

Wrapping Up

Debugging effectively saves hours of guesswork. Ray is perfect for rapid feedback during development, while Telescope gives you full visibility into requests, queries, and background jobs. Together, they provide a powerful toolkit for debugging Laravel apps across environments.

What's Next

For further monitoring and debugging, check out these guides:

Laravel Starter Kits

1v0 Ship v1.0 Faster

- Using Laravel Telescope to Debug Performance Issues
- Query Performance Tuning in Laravel + MySQL
- How to Speed Up Laravel with Database Indexing