# How do you debug a Laravel application running on a production server?

Debugging a Laravel application in production requires a cautious approach that prioritizes stability and security. The most effective methods involve using **robust logging**, specialized **error tracking services**, and the built-in `artisan tinker` command, while ensuring `APP_DEBUG` remains `false`.

## 1. Utilize Logging Effectively

Logging is your primary tool in production, as it allows you to record application activity without interrupting the user experience or exposing sensitive information.

**Log facade**: Use the `Log` facade with various severity levels (`info`, `warning`, `error`, `critical`, etc.) to log messages in your code.

**php**

```
use Illuminate\Support\Facades\Log;
Log::info('User login successful.', ['user_id' => $user->id]);
Log::error('Payment failed.', ['order_id' => $order->id]);
```

**Log channels**: Configure different log channels in `config/logging.php` to direct logs to specific destinations, such as daily files, Slack, or external services, making them easier to manage.

**Contextual information**: Always include contextual data (e.g., user ID, request details) to help pinpoint the source of an error efficiently.

## 2. Use Professional Error Monitoring Tools

For robust production monitoring, rely on specialized third-party services that provide real-time error tracking and detailed stack traces.

**Sentry**: This service offers real-time error tracking, performance monitoring, and detailed context (user info, environment, release version) for exceptions. Integrating it via the **Sentry Laravel**

**SDK** allows you to be alerted to issues before users report them.

**Bugsnag or Flare**: Other effective options that integrate well with Laravel and offer similar error monitoring and reporting capabilities.

## 3. Use `artisan tinker` for Live Inspection

The `php artisan tinker` command provides an interactive shell to interact with your application's code and data directly from the command line, which is great for testing logic without affecting the live application's front end. You can test Eloquent queries, check configuration values, and more in real-time.

## 4. Best Practices and Safety Measures

**`APP_DEBUG` must be `false`**: In your production `.env` file, ensure `APP_DEBUG` is set to `false`. Enabling debug mode in production is a major security risk that can expose sensitive information, such as your application key and database credentials.

**Avoid `dd()` and `dump()` in production**: The `dd()` (dump and die) function stops the script execution and outputs variable data, which will break your application's functionality for users. These functions are intended for local development only.

**Review logs regularly**: Access the log files in the `storage/logs` directory to review error logs and other important messages.

**Use version control**: Ensure any debugging code you add (like `Log::info()` calls) is managed through version control (e.g., Git) and properly removed or conditionalized before deploying to production

---

Revision #1
Created 9 March 2026 04:48:12 by AI Channel
Updated 9 March 2026 04:49:01 by AI Channel