Table of Contents

Introduction

If you've built custom pages in FilamentPHP, you've likely encountered a frustrating issue: your beautifully styled custom pages don't automatically adapt when users toggle between light and dark mode in the Filament dashboard. While Filament's built-in components handle theme switching elegantly, custom pages often remain stuck in light mode, creating an inconsistent user experience.

In this guide, you'll discover a simple yet powerful CSS configuration that makes your custom Filament pages fully reactive to the dashboard's theme settings. This solution requires just one line of code and works seamlessly with Tailwind CSS's dark mode utilities.

Whether you're building admin dashboards, custom reports, or specialized interfaces within Filament, this technique will ensure your pages maintain visual consistency across both theme modes.

The Problem with Custom Filament Pages

When you create custom pages in Filament using the `php artisan make:filament-page` command, you get a blank canvas to build your interface. However, these pages don't automatically inherit Filament's dark mode behavior.

Here's what typically happens:

Expected Behavior:

- User toggles dark mode in the Filament dashboard
- All components, including custom pages, switch to dark theme
- Consistent user experience across the entire application

Actual Behavior:

- Built-in Filament components switch themes correctly
- Custom pages remain in light mode
- Jarring visual inconsistency breaks the user experience

This occurs because Tailwind CSS's default dark mode implementation uses the `@media (prefers-color-scheme: dark)` query, which relies on the user's system preferences. However, Filament implements its own dark mode toggle that adds a `.dark` class to the HTML element, and your custom styles need to be configured to respect this class.

Understanding Filament's Dark Mode Implementation

Filament uses a class-based dark mode strategy rather than relying on system preferences. When a user toggles dark mode:

1. Filament adds or removes a `.dark` class to the root HTML element
2. This class triggers Tailwind's dark mode variants throughout the application
3. Built-in Filament components are pre-configured to respond to this class
4. Your custom CSS, however, needs explicit configuration to detect this change

The key insight is that Tailwind CSS needs to be told where to look for the dark mode indicator. By default, it doesn't check for a `.dark` class on parent elements, which is exactly what Filament uses.

The Solution: Custom Dark Variant

The solution is elegantly simple: we'll create a custom Tailwind variant that tells our styles to check for the `.dark` class anywhere in the parent hierarchy. This is accomplished with a single line of CSS using Tailwind's `@variant` directive.

This approach:

- Works automatically with Filament's theme toggle
- Requires no JavaScript
- Maintains full Tailwind CSS compatibility
- Supports all Tailwind utility classes
- Has zero performance impact

Step-by-Step Implementation

Step 1: Locate Your Application CSS File

First, navigate to your Filament application's CSS file. This is typically located at:

```
resources/css/filament/admin/app.css
```

Or if you're using a custom theme:

```
resources/css/filament/[your-panel-id]/theme.css
```

Step 2: Add the Custom Dark Variant

Open the CSS file and add the following line at the top, after any `@import` statements but before other custom styles:

```
@variant dark (&:where(.dark, .dark *));
```

Your complete CSS file should look something like this:

```
@import '/vendor/filament/filament/resources/css/theme.css';

@config 'tailwind.config.js';

@variant dark (&:where(.dark, .dark *));

/* Your custom styles below */
```

Step 3: Rebuild Your Assets

After adding the variant, compile your CSS assets:

```
npm run build
```

Or for development with hot reload:

```
npm run dev
```

Step 4: Test Your Implementation

1. Navigate to your custom Filament page
2. Toggle the dark mode switch in the Filament dashboard
3. Verify that your dark mode utilities now respond correctly

That's it! Your custom pages will now seamlessly switch between light and dark modes along with the rest of your Filament dashboard.

How It Works

Let's break down what the custom variant does:

```
@variant dark (&:where(.dark, .dark *));
```

`@variant dark`

- Creates a new variant called `dark` for Tailwind utilities
- Allows you to use `dark:` prefix in your classes

`&:where(.dark, .dark *)`

- `&` represents the current selector
- `:where(.dark, .dark *)` is a CSS selector that matches:
  - Elements with the `.dark` class itself
  - Any descendant of an element with the `.dark` class
- The `:where()` pseudo-class keeps specificity low, preventing conflicts

What This Achieves: When Filament adds the `.dark` class to your HTML element, any Tailwind utility with the `dark:` prefix will activate, regardless of how deeply nested your custom page components are.

Practical Examples

Example 1: Custom Dashboard Widget

Here's a custom card component that now responds to dark mode:

```
<div class="bg-white dark:bg-gray-800 rounded-lg shadow-md p-6">
    <h2 class="text-2xl font-bold text-gray-900 dark:text-white mb-4">
        Analytics Overview
    </h2>
    <p class="text-gray-600 dark:text-gray-300">
        Your dashboard content adapts seamlessly to theme changes.
    </p>
</div>
```

Result:

- Light mode: white background with dark text
- Dark mode: dark gray background with light text
- Smooth transition when toggling themes

Example 2: Custom Table Styling

Create a responsive table that matches Filament's aesthetic:

```
<table class="min-w-full divide-y divide-gray-200 dark:divide-gray-700">
  <thead class="bg-gray-50 dark:bg-gray-800">
    <tr>
      <th class="px-6 py-3 text-left text-xs font-medium text-gray-500 dark:text-gray-400 uppercase tracking-wider">
        Name
      </th>
      <th class="px-6 py-3 text-left text-xs font-medium text-gray-500 dark:text-gray-400 uppercase tracking-wider">
        Status
      </th>
    </tr>
  </thead>
  <tbody class="bg-white dark:bg-gray-900 divide-y divide-gray-200 dark:divide-gray-800">
    <tr>
      <td class="px-6 py-4 text-sm text-gray-900 dark:text-gray-100">
        John Doe
      </td>
      <td class="px-6 py-4 text-sm text-gray-600 dark:text-gray-400">
        Active
      </td>
    </tr>
  </tbody>
</table>
```

Example 3: Custom Form Elements

Build forms that integrate perfectly with Filament's theme:

```
<form class="space-y-4">
  <div>
    <label class="block text-sm font-medium text-gray-700 dark:text-gray-300 mb-2">
      Project Name
    </label>
    <input
      type="text"
      class="w-full px-4 py-2 border border-gray-300 dark:border-gray-600 rounded-lg
             bg-white dark:bg-gray-800 text-gray-900 dark:text-white
             focus:ring-2 focus:ring-primary-500 dark:focus:ring-primary-400"
      placeholder="Enter project name"
    />
  </div>

  <button
    type="submit"
    class="px-6 py-2 bg-primary-600 hover:bg-primary-700 dark:bg-primary-500 dark:hover:bg-primary-600
           text-white rounded-lg transition-colors"
  >
    Create Project
  </button>
</form>
```

Example 4: Alert Components

Create attention-grabbing alerts that work in both themes:

```
<div class="bg-blue-50 dark:bg-blue-900/20 border-l-4 border-blue-500 dark:border-blue-400 p-4 mb-4">
  <div class="flex">
    <div class="flex-shrink-0">
      <svg class="h-5 w-5 text-blue-500 dark:text-blue-400" fill="currentColor" viewBox="0 0 20 20">
        <path fill-rule="evenodd" d="M18 10a8 8 0 11-16 0 8 8 0 0116 0zm-7-4a1 1 0 11-2 0 1 1 0 012 0zM9 9a1 1 0 0
      </svg>
    </div>
    <div class="ml-3">
      <p class="text-sm text-blue-700 dark:text-blue-300">
        This is an informational message that adapts to your theme preference.
      </p>
    </div>
  </div>
</div>
```

Best Practices and Tips

1. Use Semantic Color Naming

Instead of hardcoding specific shades, use Tailwind's semantic color scale:

```
<!-- Good: Scales naturally between themes -->
<div class="bg-gray-100 dark:bg-gray-800">

<!-- Avoid: Doesn't provide good contrast -->
<div class="bg-gray-100 dark:bg-gray-200">
```

2. Maintain Sufficient Contrast

Always ensure text remains readable in both modes:

```
<!-- Good contrast in both modes -->
<p class="text-gray-900 dark:text-gray-100">

<!-- Poor contrast in dark mode -->
<p class="text-gray-700 dark:text-gray-600">
```

3. Test Both Themes During Development

Keep your browser's developer tools open and toggle between themes frequently. What looks perfect in light mode might be illegible in dark mode.

4. Leverage Filament's Color Palette

Filament provides a carefully crafted color palette. Use primary, success, warning, and danger colors that are already optimized for both themes:

```
<button class="bg-primary-600 hover:bg-primary-700 dark:bg-primary-500 dark:hover:bg-primary-600">
    Primary Action
</button>
```

5. Consider Shadows and Borders

Shadows and borders behave differently in dark mode:

```
<!-- Adaptive shadow and border -->
<div class="shadow-md dark:shadow-lg border border-gray-200 dark:border-gray-700">
    Content with proper depth in both themes
</div>
```

6. Use Transition Classes

Add smooth transitions when switching themes for a polished experience:

```
<div class="transition-colors duration-200 bg-white dark:bg-gray-800">
    Smooth theme transitions
</div>
```

For more tips on building professional Filament applications, visit cherradix.dev for comprehensive tutorials and best practices.

Troubleshooting Common Issues

Issue 1: Dark Mode Not Working After Adding Variant

Symptom: Dark mode utilities still don't respond to theme toggle.

Solution:

1. Ensure you've rebuilt your assets: `npm run build`
2. Clear your browser cache (Ctrl+Shift+R or Cmd+Shift+R)
3. Verify the variant is in the correct CSS file that Filament is loading
4. Check that there are no syntax errors in your CSS file

Issue 2: Styles Working in Dev but Not Production

Symptom: Dark mode works with `npm run dev` but not after deployment.

Solution:

1. Run `npm run build` before deploying
2. Ensure your build process includes the CSS compilation step
3. Verify that the compiled CSS file is being deployed to your server
4. Check your asset versioning and cache busting strategy

Issue 3: Inconsistent Behavior Across Pages

Symptom: Some pages respond to dark mode, others don't.

Solution:

1. Verify all custom pages are using the same CSS file
2. Check that you haven't imported different CSS files in different views
3. Ensure the variant is added before any custom styles that might override it

Issue 4: Variant Not Recognized by Tailwind

Symptom: `Unknown variant "dark"` error or dark utilities not generating.

Solution:

1. Ensure you're using a Tailwind CSS version that supports the `@variant` directive (v3.1+)
2. Check that the syntax is correct: `@variant dark (&:where(.dark, .dark *));`
3. Verify the `@config` directive points to your correct Tailwind config file

Issue 5: Performance Issues with Dark Mode

Symptom: Lag when toggling between themes.

Solution:

1. The custom variant itself has no performance impact
2. If you experience lag, it's likely from too many transition effects
3. Reduce the number of elements with `transition-colors` or adjust duration values

Issue 6: Specificity Conflicts

Symptom: Dark mode styles not overriding default styles.

Solution: The `:where()` pseudo-class in the variant keeps specificity low by design. If you need higher specificity:

```
/* Alternative with higher specificity */
@variant dark (.dark &);
```

However, this may cause other specificity issues. The `:where()` approach is recommended for most cases.

Conclusion

Making your custom Filament pages reactive to dark and light mode doesn't have to be complicated. With a single line of CSS configuration, you can ensure that your entire application maintains visual consistency when users toggle between themes.

Key Takeaways:

1. **Simple Implementation**: Just add `@variant dark (&:where(.dark, .dark *));` to your CSS file
2. **No JavaScript Required**: This is a pure CSS solution that works automatically
3. **Full Tailwind Integration**: Use all of Tailwind's utilities with the `dark:` prefix
4. **Zero Performance Impact**: The variant has no runtime performance cost
5. **Consistent User Experience**: Your custom pages will feel native to Filament

This technique transforms your custom Filament pages from theme-ignorant to theme-aware, creating a polished, professional experience that your users will appreciate. Whether you're building complex dashboards, custom reports, or specialized admin interfaces, this approach ensures your application looks great in any theme.

Remember to test both light and dark modes throughout your development process, maintain proper contrast ratios, and leverage Filament's existing color palette for the best results.

Ready to level up your Filament development skills? Visit cherradix.dev for more in-depth tutorials, tips, and best practices for building exceptional Laravel and Filament applications. From authentication systems to performance optimization, we've got you covered with practical, production-ready solutions.

Happy coding, and may your dark mode always be pixel-perfect!