Ingo Steinke
Posted on May 23 • Updated on May 27

Classic Themes with Block Patterns in WordPress

#wordpress #webdev #tutorial

Using WordPress as a Developer (10 Part Series)

1 Did I upgrade WordPress to PHP 8 too early?

2 Don't Update WordPress Plugins ...

... 6 more parts...

9 **Classic Themes with Block Patterns in WordPress**

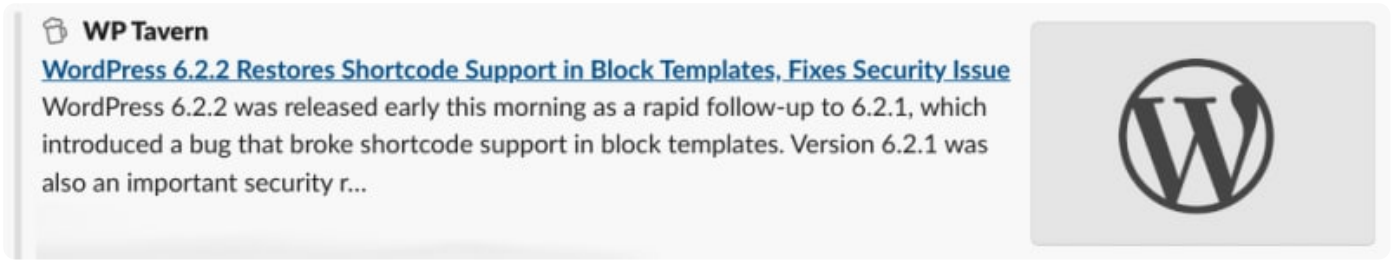
10 Filter WP Post Frontpages in the Loop with Polylang

This post about the WHY and HOW to develop classic themes in 2023 rounds up my DEV blog series about WordPress.

Why Classic Themes: 20 Years of History

WordPress is celebrating its 20th anniversary and it still powers millions of websites all over the world, so we can say that's an open-source software success story. The "WordPress multiverse" has been fragmented for a long time, due to a multitude of possible combinations of plugins and page builders like Elementor. But efforts to unify mainstream development and obsolete page builders by offering full-site editing in the Gutenberg block editor has caused controversy. While no-code enthusiasts are happy about the new features, many frontend developers have been frustrated about breaking changes and unfinished development, rolled out and promoted by WP as the new default as part of their "blogs to blocks" campaign.

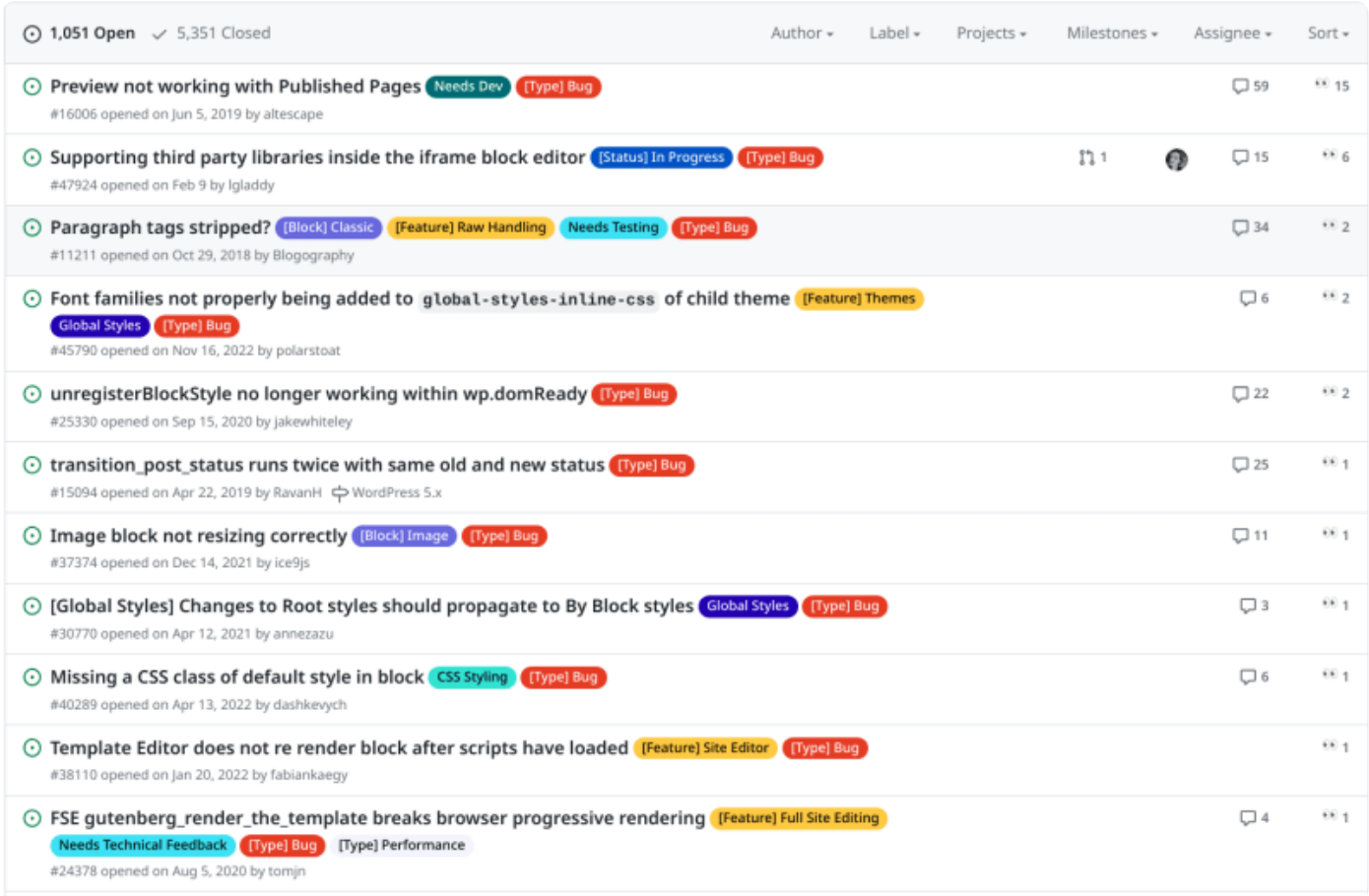
Recently, WordPress 6.2 introduced a change that disabled shortcode functionality in existing block themes, a (security) (mis)feature rolled back as part of the 6.2.2 bugfix update. Currently, the Gutenberg project has nearly 5000 open issues, including more than 1000 open bug tickets on GitHub.



The Gutenberg project is based on good intentions, but even when it works, look at the markup it generates! Why would any "new" software use so many inline styles in 2023, alongside a theme engine and a lot of utility classes?

Unlike Elementor's individual class names (on the left), Gutenberg applies inline styles, making it hard to impossible to override using custom CSS without using `!important`.

1K Open Bug Reports and Gutenberg's many Issues



WordPress 6.2 “Dolphy”, the first major release in 2023, includes more than 900 enhancements and bug fixes. But WordPress/gutenberg still had 4842 open (and 18128 closed) issues at the time of writing, including 1051 open issues labeled as bugs.

Classic WordPress Themes in 2023 and beyond

Luckily, we can still use and develop classic themes in 2023, using the block editor or the classic editor, or both – depending on the content type.

There are full site editing enabled block themes, block themes, hybrid themes, and classic themes to chose from. So we don't need to go all in and use all the new features, unless we want to.

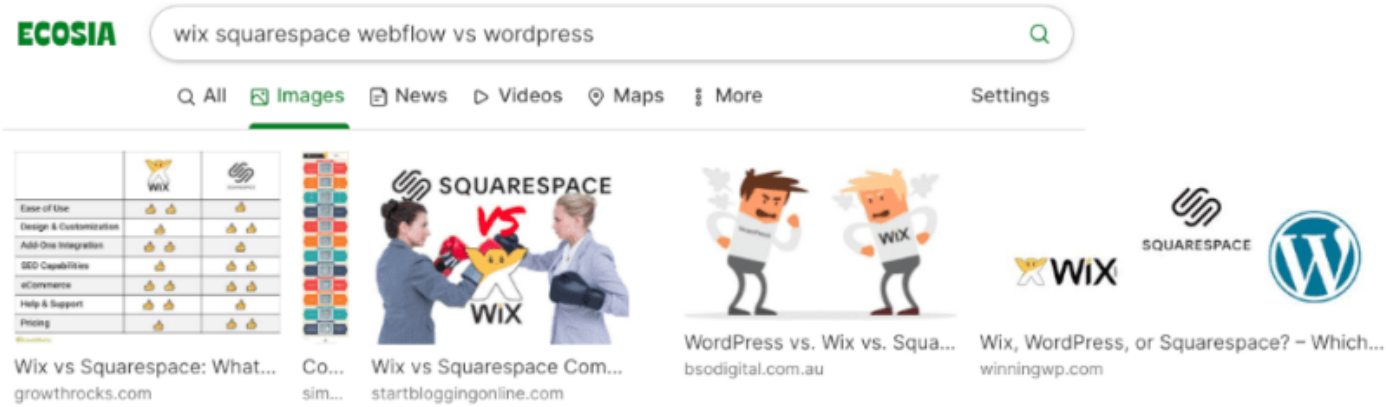
WordPress isn't in danger (yet)

I doubt that WordPress is in danger. Its multiverse / flexibility and ease of use compared to real content management systems like Typo3, NEOS or Drupal, and the ability to just install it and control your content on your own webserver, try that with Wix, WebFlow, SquareSpace, Shopify and all those "serverless" software as a service servers. Hopefully, the core team will either get their block editor right some day (WordPress 7.0?) or make it completely optional. That would still be better than a fork,

but I'd wished they had improved **security**, **performance** and **internationalization** instead of releasing the new editing features in an unstable beta state.

Reasons to stick to WordPress

I am still looking for a content management system that allows people without coding skills and without a technological mindset to edit their own content with some kind of "what you see is what you get" preview (no, they do not want to edit Markdown files, and they don't want to learn the WebFlow UI either) but have more creative freedom and more professional results than construction kits like Wix or SquareSpace seem to offer.



Despite its cluttered markup and quirky validation restrictions, the block editor can provide quite helpful for content creators, and we don't need to go along with all that it offers, as there are some pragmatic approaches like block groups, block patterns / templates, and block variations, that we can provide without having to develop custom blocks.

WordPress Development without Losing my Head

Trying to work against the intended way can lead to a lot of problems and frustration. Many WordPress developers seem to stick to a routine and reuse their templates and workflows to make their work scale effectively, also from a financial point of view. I also tried to find my own way to mix the new features with classic ones. As we saw, shortcodes still work (again, since WordPress 6.2.2), at least inside shortcode blocks. I will also show that we can practice modern web development, as it is known outside the WP bubble, and develop classic themes without following outdated classic coding standards.

How to develop?

Classic Themes with Optional Block Features

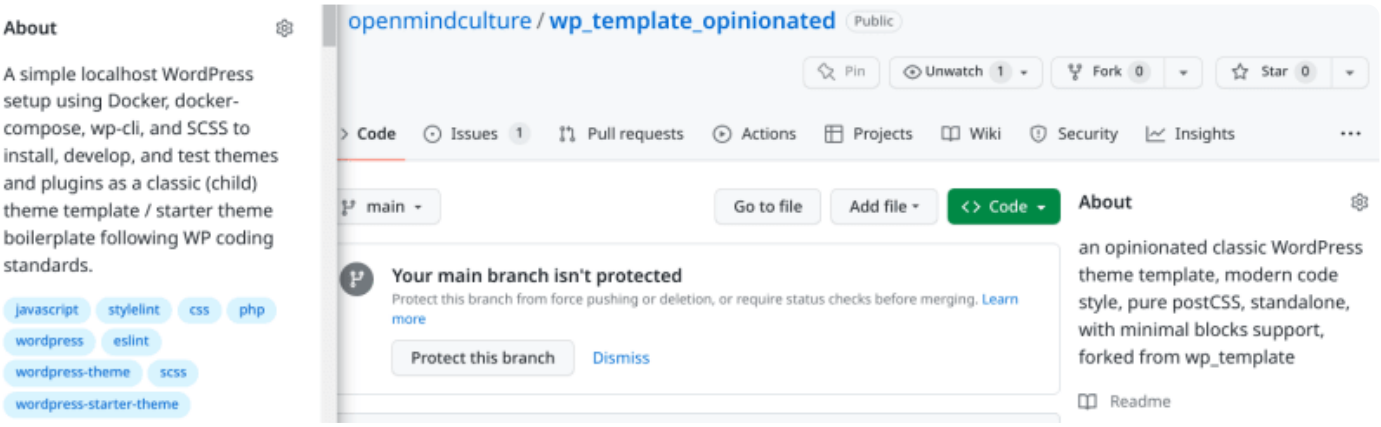
We can use block editor features without giving up control and without letting our website turn into a bloated single page application full of redundant inline styles. This classic approach is not even considered a "hybrid" theme, as we don't use full site editing, no style engine (`theme.json`), and we won't be building custom blocks either.

Revisiting WordPress after working with React, JamStack, and CSS-centered design systems, it can take some time to figure out the best way to achieve maintainability, accessibility, and a good user experience using modern WordPress themes. Let me show you a website that uses WordPress with its latest editor features and still achieves top scores on Google's PageSpeed Insights, WAVE, SecurityHeaders, EcoGrader, WebsiteCarbon, etc.

My Opinionated WordPress Theme Boilerplate

I don't want to copy an existing website to start developing the next one. So I have started a more abstract repository as a future development template instead.

A side product of my research and development, [wp_template_opinionated]() is my own WordPress starter theme boilerplate. It includes several aspects of what I found to be a useful project setup close enough to popular development trends and official best practice.



Custom Fields to Avoid Block Editor Issues

While `wp_template` is focused on providing a simple localhost theme development setup, the setup for the Berlin fashion website has been modified to include some

more specific styles and components. I did not use any block features for the fashion website. Instead, custom post types (CPT) and (advanced) custom fields(ACF) with minimal WYSIYWG functionality to add links and basic formatting like bold or italic text. A defined (but extensible) content structure with limited editability should provide enough freedom to the content creator without risking to destroy the website layout due to the block editor's many issues and misfeatures.

Client-Side or Server-Side Rendering?

But even a completely finished, bug-free, block theme would be no perfect solution for every kind of user. Often, there is more than one way to do things. Block themes tend to put logic in the front-end, turning a website into a web application. This can have some advantages, like lazy loading. On the other hand, the classic server-side site generation produces static web pages than can be cached and served quickly.

Child Theme or no Child Theme?

The best practice for a quick, minimal, and standards compliant setup, is to create a child theme of an official or popular theme that provides most of the required functionality. A parent theme might look nice at first sight, but introduce limitations or paradigms that don't go along well with our needs or preferences.

Disadvantages of Child Themes

The disadvantages of child themes increase, the more we want to develop our own design and layout and start to work against the original styles and assumptions. This becomes even worse when using default blocks: now we have to make sure that we override our parent theme's styles as well as the default WordPress core blocks styles.

```
.wp-block-buttons {
    margin-left: 0 !important; // override unhelpful defaults of parent theme or WordPress core?
}
```

Advantages of Child Themes

On the other hand, a good parent theme can spare us a lot of boilerplate code, mistakes and missing features, especially one that has been optimized to be quick, responsive, and accessible.

The most obvious option would be the latest official standard themes, like Twenty Twenty-Three. But the latest official WordPress themes support full-site editing and the new theme engine, providing a lot of end-user options that might get in our way. As an alternative, the popular GeneratePress aims to be a lightweight foundation for theme development, and its optional extension are shipped seperately, like the GenerateBlocks plugin that provides a small collection of lightweight blocks optimized but not limited to be used with GeneratePress or a child theme.

As you might have guessed, even the "minimal" markup produced by GeneratePress, even more so when combined with WP Core class names and Gutenberg inline styles, exceeds what I consider to be good old HTML. This is why I ended up coding a classic theme "from scratch" (obviously copy-pasting from here and there). We will see more details and examples below.

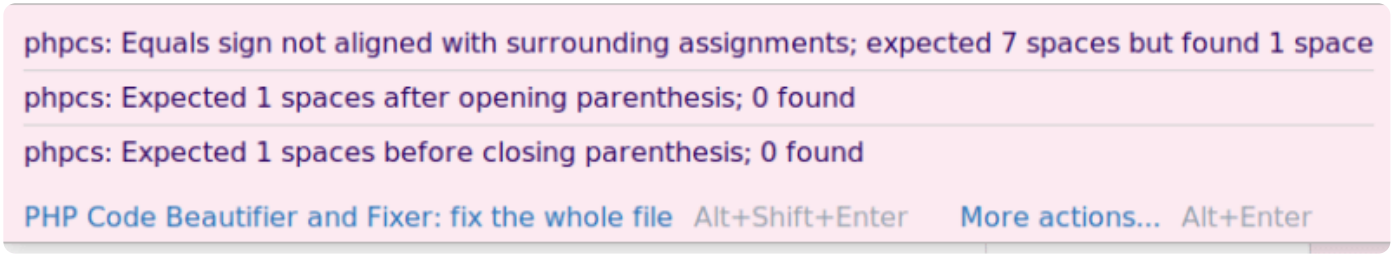
Minimal Setup for any Kind of Theme

My personal theme development template can be used both for developing child themes, as well as for starting from scratch. This article is partially based on the template's documentation. You can check for a more recent update in the GitHub repository at github.com/openmindculture/wp_template#readme.

The template repository has been forked from wp_cli_docker to help us build a classic (child) theme from scratch (without exporting from the block editor), that follows the official WordPress coding standards. We can still support and provide custom blocks in our theme.

Avoiding Ancient Coding Standards

A note on WordPress Coding Standards. While they are mandatory for developing code to be released in the official plugin directory, some of the rules are outdated and contradict the current best practices specified in PSR-12 (PHP Coding Style) and eslint-config-recommended (JavaScript / ES2023).



Letting PHP Code sniffer suggest the official WordPress syntax without turning off other code quality tools like SonarLint will clutter your editor with contradictory suggestions to add or remove spaces inside paranthesis or to align or not align equals signs with surrounding assignments.

Abide to the Ancient Standard when Coding Official Plugins!

I once regretted my personal preference to ignore the old coding styles when submitting a plugin for review, but otherwise I would rather adhere to modern standards.

Coming back to the main topic, I also tend to use some of the latest, modern block editor features, either with or without using the theme engine, which is the point of this this article.

How to disable Full Site Editing programmatically?

I have tried to disable full site editing programmatically. WordPress 6.2 seems to show site editing features instead of the customizer in its WP-Admin side menu, while still linking to the classic customizer from the top bar.

Again, why waste our energy and work against an ever changing UI unless it's really important? There is a lot that we can do without even caring about what's enabled or not.

Like the block inserter which still offers an additional tab for custom content, reusable blocks, and blocks of blocks, offering an alternative method for complex components than extending or building theme blocks with custom callbacks etc.

Blocks of Blocks: Block Groups

Known as "block groups" or blocks inside a group block, we might also describe them as higher order blocks: any content can be saved as a block that appears in our block library, and so can a group of existing blocks.

We can use the block editor UI to configure some basic blocks, and use our theme to provide the code as ready-made theme-blocks that can be used and modified by

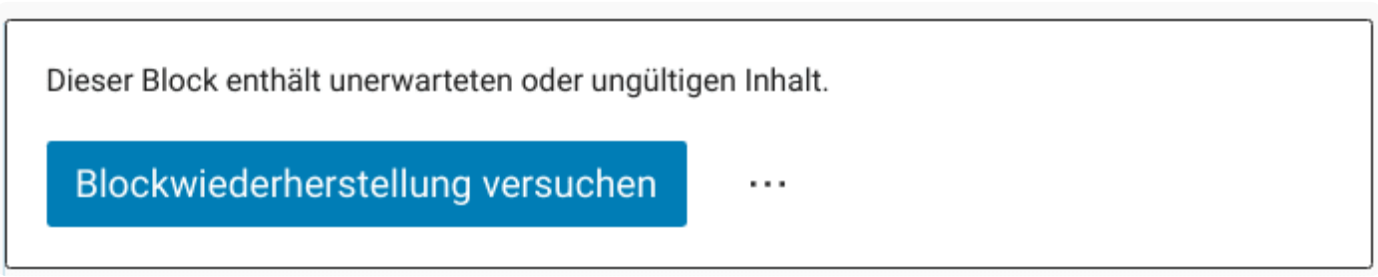
content creators. We don't necessarily need to create configurable custom blocks.

Persisting Higher-Order Blocks as PHP Patterns

We can save block groups source code to PHP pattern files to provide block patterns in the template library. Unfortunately, I still have not found a way to validate the exported code. Even with every recommended WordPress coding standard configuration and PhpStorm WordPress assistance, sometimes the code validates as 100% correct, but still fails to satisfy the block editor and its arcane, mostly undocumented syntax engine.

The Infamous Block Recovery Message

In the best case, we will see our pattern in the library, but get the infamous "block recovery unexpected or invalid content", with no details like line number.



But it can get worse with more severe errors that make WordPress silently ignore our template file. In which case we should hopefully see a warning in our IDE.

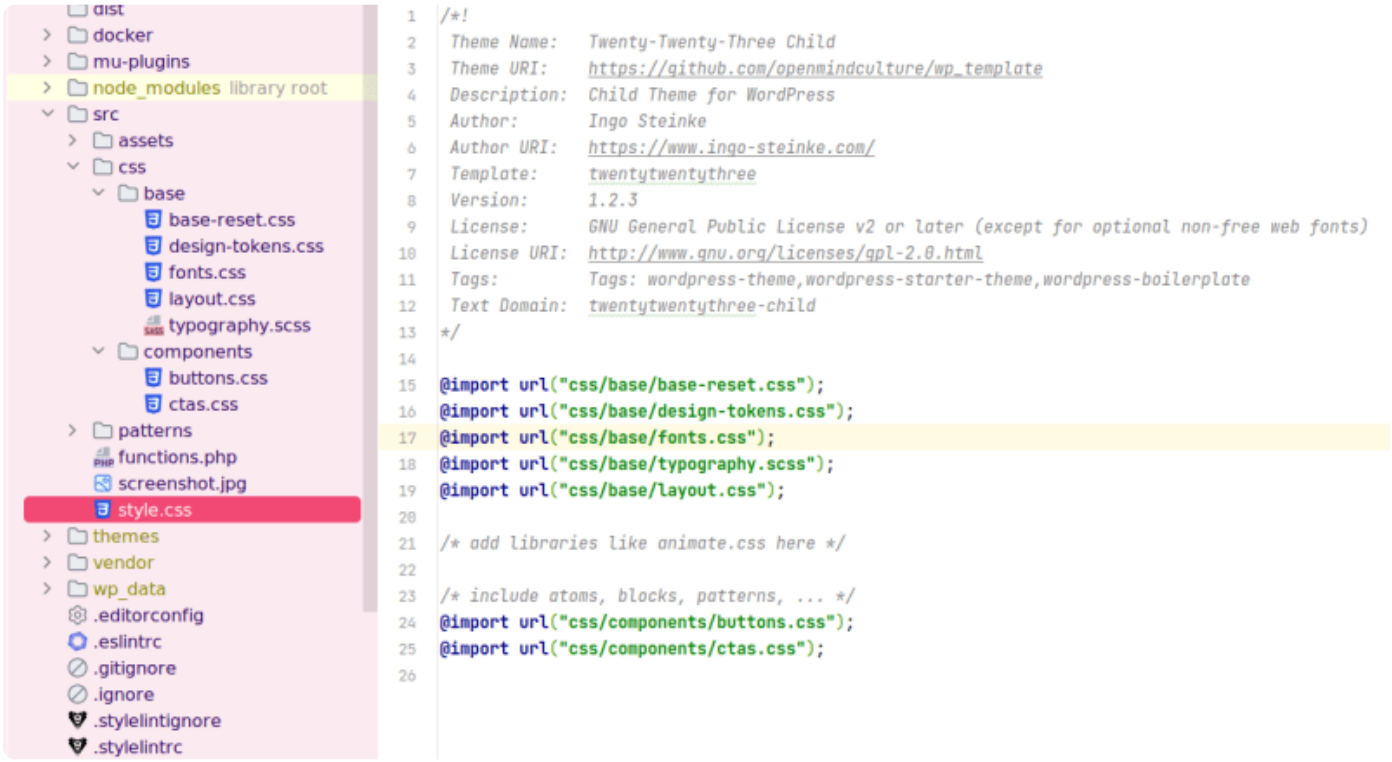
Mandatory Must-Use-Plugins for Separation of Concerns

If we need to define custom data structures, we can use the popular ACF (Advanced Custom Fields) plugin to provide custom fields, and use our own minimal plugin to define custom post types. This should not be part of the theme, to prevent data loss when a theme is changed or deactivated. Using a must-use plugin ensures that the code will always be active. Custom fields defined using the ACF user interface in WordPress admin can be exported to JSON or PHP, so we can add it to our custom fields plugin as well.

To persist important content like example posts or pages, use the default WordPress exporter to save an XML export as `content.xml`, which will automatically be imported when setting up the local environment using `npm install`. The data might have to be installed manually when setting up a production environment.

PostCSS, SCSS, and optional TypeScript Support

For the sake of simplicity, my example setup uses a single-file plugin and puts all styles directly into a single `style.css` file without using further `theme.css` or `theme.json` files, which we might want to use depending on the requirements for customizability. Likewise, SASS / SCSS support can be added if it makes life easier for the developer(s) involved. But as we already use PostCSS to control autoprefixing and cssnano minification, we can also use it to support the latest and even upcoming CSS sytax like native CSS nesting.



TypeScript / ECMAScript transpilation can also be configured, which might be especially useful for websites / web applications with a lot of client-side functionality. As the original use-cases of my template setup was focused on CSS development, in the current state at the time of writing, any JavaScript code will be exported exactly as it has been written.

Coding Standards and Linter Rules

I have relaxed some linter rules, deviating from the original WordPress coding standards, but inspired by changes introduced by the Gutenberg team, and deviating from ESLint defaults to allow more compatible, ES3-style JavaScript using `var` instead of `let` and `const`.

Apart from those changes, my original setup still tries to comply with the current WordPress development style. (The opinionated version does not care anymore.) We can use any tooling and code assistance available, to compensate for the absence of a

complete a proper testing and validation pipeline in the WordPress developer ecosystem.

Code Inspection 🕵️

To facilitate debugging, `plugins` is mounted as a local directory, so you can search files and view error messages and annotations, to collect details for filing issues or for creating patches yourself.

You can enter the Docker container and use it much like a remote server.

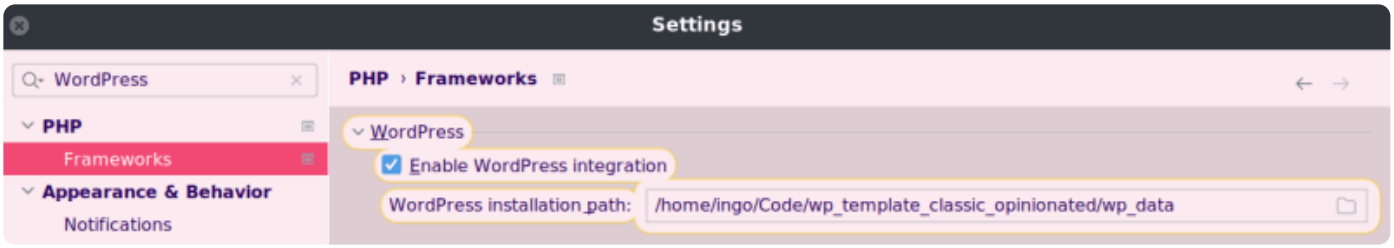
Type

```
docker exec -it wordpress_setup_test_wordpress_1 bash
```

Replace `wp_cli_docker_wordpress_1` with the appropriate name found in `docker ps`, if necessary.

Inside the docker container, you will find WordPress in `/var/www/html`.

In JetBrains IDEA (PhpStorm, WebStorm, etc.) enable WordPress support and set `wp_data` as WordPress path, so that the local code inspections like SonarLint can recognize the built-in functions. You can still mark the directory as excluded to avoid unnecessary indexing and search results.



Some SonarLint warnings (and PHP PSR conventions) should be ignored, like avoiding underscores in function names. As we operate in a global namespace shared with other plugins, it is considered best practice to use a unique prefix for identifiers used for `function`, `class`, and `define`.

A local code sniffer validation can be set up using the provided `composer.json` configuration. Note that this currently does not work with PHP 8, so you need to use a PHP 7.4 runtime (`/usr/bin/php74`). You may need to adjust the IDE settings to WordPress coding standards and code sniffer configurations according to the provided tutorial.

- https://packagist.org/packages/wp-coding-standards/wpcs

- https://www.jetbrains.com/help/idea/using-php-code-sniffer.html#installing-configuring-code-sniffer
- https://www.linuxbabe.com/ubuntu/php-multiple-versions-ubuntu

As I mentioned earlier, we might decide to ignore the WordPress coding style completely and follow common coding standards for PHP, ECMAScript, and CSS instead, as long as we do not plan to ever release our themes and plugins in the official directories.

Child Themes and Editor Styles Specificity

As mentioned in the child-theme chapter of the WordPress theme developer handbook, different themes might use different ways to load their theme styles and editor styles, so we might have to adapt our loading mechanism accordingly.

Some themes automatically load child theme styles, so that we would not need any custom `functions.php` code. But we can still use a `wp_enqueue_scripts` hook with two `wp_enqueue_style` function calls to state the styles' dependency explicitly.

To apply our (child) theme styles in the (Gutenberg) block editor, we have to `add_theme_support` for `editor-styles` and add our `theme.css` using `add_editor_style` as described in the block editor handbook. This will wrap our frontend styles in an `.editor-styles-wrapper` which is the modern equivant of the class editor's iframe body.

It is still a good idea to ensure that our child theme styles override parent styles using CSS specificity, as suggested by GeneratePress support. This makes our theme more robust as it will not have to rely on the loading order.

Note that a `body` selector will be automatically replaced by `.editor-styles-wrapper` which would have been added anyway, so no higher specificity in the backend, and that the semantic landmark elements `main`, `article` and `header` are missing inside the block editor preview, and there is no `.site` There is an `.editor-styles-wrapper .wp-block-post-content`, but we would not want to add backend-specific selectors to our frontend styles. Using a `body div` wrapper is not the most elegant workaround either.

In case that we only need the markup generated by the parent theme, we could dequeue the parent the style and omit any specificity wrappers altogether.

Which WordPress / Plugin Class Names to keep?

Even if we remove and replace inline styles and core class names with our own (A)BEM / ITCSS terminology, there are some exceptions where it might be helpful to keep "official" class names: images / media library, and any markup generated by plugins.

As we decided to use WordPress for reasons like content management, communication, and spam protection, we can make use of that functionality. My recent classic theme supports image sizing and alignment classes, as well as everything emitted by Contact Form 7, Complianz GDPR privacy / cookie compliance, Polylang internationalization, and header / footer widgets including the mobile "burger" menu. We may also consider your favorites and popular "must-have" plugins like Yoast and JetPack.

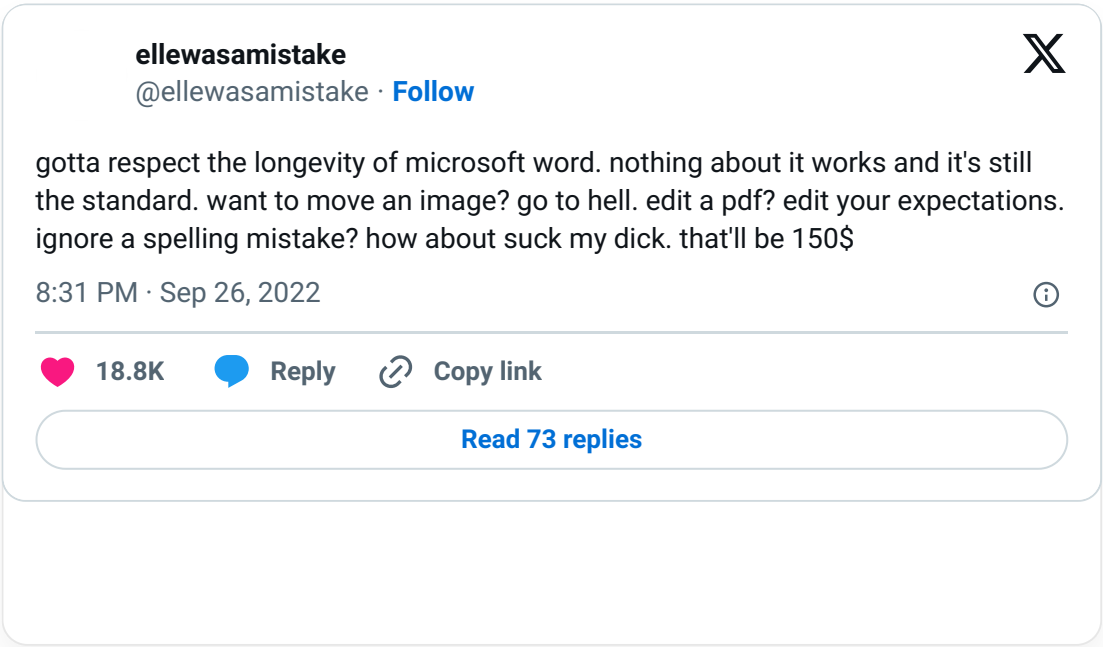
In future themes, we might want to support the new WordPress style engine and use `theme.json`, but we don't necessarily need to. See Managing CSS Styles in a WordPress Block Theme by Ganesh Dahal on CSS-Tricks for more details.

Conclusion: Classic Theme Development is still relevant and appropriate in 2023

I hope that you have found some useful advice in this article series about WordPress development. Hopefully, this will be my last post about WordPress. I will be looking into other site generators and frameworks like NEOS and Symfony, and continue my ongoing JAMstack and MERN side projects in 2023, and also continue to explore and blog about new and upcoming HTML, Vanilla JS and CSS features.

As I said, I have been using WordPress as a developer for a very long time. I am not a fan of the new features starting with Gutenberg, but I also wanted to find a way to use it properly, if I have to. This post rounds up my blog post series sharing my experience with other developers.

After 20 years, WordPress seems to become a living legend like Word:



> gotta respect the longevity of microsoft word.
> nothing about it works and it's still the standard.

To be fair, both Word and WordPress have become popular for a reason, and there is a lot that does work, at least somehow, for so many users with so many different use cases.

Some users love the new block editing features, some don't, and that's fine unless someone else tries to decide about the best and only path that everyone must follow.

Classic WordPress Themes aren't going away...

My personal takeaway is that it's totally okay to stick to classic themes without using site builders, full site editing, or the WordPress theme engine. We can still use some of the latest WordPress features, and we can prevent our classic themes from making the site look broken in the block editor.

Another takeaway: it is not necessary to develop a headless CMS front-end just because you might have the same opinion about Gutenberg as I do.

Like speckyboy and other fellow developers concluded recently, classic WordPress themes aren't going away, at least not in 2023, and probably not in 2024 either.

P.S. (added an 27 May) in a recent discussion on Twitter, where Derek Ashauer asked if "FSE is solving a problem that doesn't exist", I learned that there is a classic WordPress fork without the block editor: ClassicPress aims to be a "lightweight, stable, PHP-based CMS for creators."

Derek Ashauer
@DerekAshauer · **Follow**

Is FSE solving a problem that doesn't exist? Most beginners or single site owners don't want that much access, it's too overwhelming - they are happy with a predesigned theme + basic options. Pros already have more advanced tools available. Or FSE just still has long way to go?

> **Jamie Pootlepress - YouTuber and WP Plugin builder** @pootlepress
> When I run my Beginners WordPress courses I give students a quick 10 minute demo of the wp customizer and within a few minutes they are happily making theme changes. IMO for Block Themes to increase their current low download rate, the Site Editing experience needs to be as easy.…

3:03 PM · May 26, 2023

21 Reply Copy link

Read 13 replies

Using WordPress as a Developer (10 Part Series)

1	Did I upgrade WordPress to PHP 8 too early?
2	Don't Update WordPress Plugins ...
...	6 more parts...
9	**Classic Themes with Block Patterns in WordPress**
10	Filter WP Post Frontpages in the Loop with Polylang

Top comments (4)

R **Rimsha Victor Gill** • Jun 23

I appreciate the valuable content provided here. I've been exploring the realm of developing WordPress themes from scratch. Thank you for shedding light on this matter.

WordPress continues to evolve, offering a pragmatic approach to theme development in 2023. Developers can still embrace classic themes while selectively utilizing the block editor's features. By following best practices, leveraging plugins, and choosing the right tools, they can create maintainable, accessible, and high-performing websites using WordPress.

David Itam Essien • Jun 18

@ingosteinke this is wonderful content. Thank you. I have been looking into wordpress theme development from scratch myself and have not been able to find my way around it.

I want to start from scratch in order to enable me understand the whole process and components I am working with. Please can you point me to resources I might use; Up-to-date resources.

Ingo Steinke • Jun 29

If you want to understand how (classic) themes work, I would rather recommend to start with a child theme of a classic theme first, and build one from scratch afterwards, copy-pasting from the last project's parent theme step by step but not without understanding. Hesitantly you will learn enough to start from scratch. Unfortunately I am not sure which up-to date tutorials are the best to start from scratch, but on wordpress.org, there is a lot of good documentation, mostly based on classic theme development.

Ingo Steinke • Aug 16

Pragmatic development: after making it clear how I feel about theme.json and the block editor, I turned around 180 degrees and set up wp_block_theme_child, a template for a very simple Twenty Twenty-Three child theme with block support and full site editing, ready to use with my WordPress docker setup, for the occasional small business owners who don't need much customization.

Now I am officially 100% unopinionated, at least when it comes to what kind of WordPress theme I can build for you. Elementor, Gutenberg, or classic / hybrid - you name it! But of course I am not unopinionated at all and I still prefer classic themes with custom fields and custom post types.