Gutenberg: A step forward or two steps back?

I got into the discussion for an issue in the _s theme repository regarding Gutenberg code blocks. It seems that after the first phase where they will be optional, meaning that if you don't want them or have content that relies on the traditional editor's layout you can choose not to use them or disable them via another plugin, you will be forced to use Gutenberg and its blocks regardless of what you needs are.

The assumption that making it easier for everyone to create Wordpress content using Gutenberg blocks is not just misguided but dangerous too without providing an alternative way to create the content we'll display to our users.

Over the past ten years I've built workflows and blogs with Wordpress that do pretty much everything and anything I need them to without having to largely modify themes or templates. For example: I use Jetpack's Markdown with a customized local installation of the Prism highlighter (rather than a plugin) for content creation (including working around a bug in Jetpack's Markdown implementation that seems to be too low priority for the Jetpack team to fix). Rather than use shortcodes I've become used to writing straight HTML that is styled to display the way I want it.

Gutenberg uses the concept of blocks, smaller self-contained units that allow for discrete portions of rich, visual content as part of your Wordpress posts and pages.

> The ship, like Theseus', needs to continue sailing while we upgrade the materials that make it. It needs to adapt to welcome new people, those that find it too rough to climb on board, too slippery a surface, too unwelcoming a sight, while retaining its essence of liberty. This is not an easy challenge—not in the slightest. Indeed, we called it Gutenberg for a reason, for both its challenges and opportunities, for what it can represent in terms of continuity and change. It is an ambitious project and it needs the whole WordPress community to succeed. I'd like to start examining some of the decisions made along the way, and the ones that are still to

> come.
>
> From Gutenberg, or the Ship of Theseus

The goals are admirable but the execution is where I'm having issues. I don't think that forcing everyone into a new paradigm that doesn't include what has been done in the past is a good long-term solution.

What worries me the most is that there is no real migration path for existing themes and content because, in my opinion, the emphasis has been made on new users and ease of authoring for them. There is no consideration to what it'll take to update existing content and how much work it'll take to migrate the older projects.

As with everything in Wordpress, the trend seems to be that we need to make everything as simple as possible without regard for people who may not want that level of simplicity, you're stuck with it regardless. Other than the issue discussed below there has been no mention about a plan not to break customized installations of Wordpress or existing workflows that don't use (or need) visual editing.

In the earlier stages (I believe it'll start with the 5.0 release) you will be able to disable Gutenberg through another plugin but as the project moves forward the Gutenberg will become so baked into Wordpress core and the Wordpress "way of doing things" that disabling it will become impossible.

I've tried using my existing workflow in a Gutenberg-enabled clone of my existing site and I've encountered many problems that make me not want to use Gutenberg at all. I'll detail some of the issues later.

There is an issue in the Github repository that discusses Viable Migration Plan for Minimum Viable Product that began the discussion of what it would take to keep the current editor and Gutenberg so that existing installations will not break if Gutenberg is active.

Target Audience and Expected Expertise

> On Guides and Placeholders. It is true that WordPress is capable of creating sophisticated sites and layouts. The issue is

> you need to know how to do it. And getting there requires a level of commitment and expertise that should be unnecessary. Why do users have to bear the weight of convoluted setups to work around the lack of a solid and intuitive visual experience?
>
> This question is what brought us to the concept of blocks in the first place. The simplified nature of blocks, which optimizes for the direct manipulation of content and a single way to insert it, proposes an evolution of the WordPress model. It also comes with interesting opportunities. For example: how many themes have remarkable presentation in their demo sites but require Herculean efforts to replicate? A core ingredient of blocks is that, by their nature, they have a defined "empty state" that works as a placeholder describing how someone can interact with it. Blocks can guide a user as they craft their content intuitively.
>
> From Gutenberg, or the Ship of Theseus

The same argument can be made for the reverse case. We've been using customized versions of TinyMCE as far back as I've been using Wordpress (since version 2.6 in 2006) and people have learned to use it, code and work around its (many) limitations and have produced awesome content... only for Wordpress to come and tell you to start over both as a developer, a designer and a user.

Biggest question, so far, is this: **Who is Gutenberg really for?**

I ask because in reading posts and documentation for Gutenberg seems to confuse who will use and benefit the most of Gutenberg and its functionality.

If I understand it correctly the idea behind Gutenberg is that it'll take over from plugins and shortcodes and you'll get a series of boxes that are purpose built for a single function and that users will compose posts and pages for.

But then the idea of building custom blocks is where I'm getting confused. Are the blocks part of core, part of of a theme, both or neither? How will they work with Wordpress Child Themes?

They seem to be part of core, at least the default blocks, themes and plugins (in the sense that themes can add and restrict what blocks you have available to

use)

So, if I'm understanding this correctly, blocks take the place of plugins and shortcodes but there is no clear or easy way to figure out what will be supported through blocks or what older pieces of content will work as they are and what will we need to migrate or create new blocks for.

Markdown as structure or content authoring?

> Markdown can often be great for writing, but it's not necessarily the best environment for working with rich media and embeds. With the granularity afforded by blocks you could intermix markdown blocks with any other block, adapting to whatever is the most convenient way to express a specific kind of content. Whenever a block cannot be interpreted, we can also handle it as a plain HTML block. In future releases, you'd be able to edit individual blocks as HTML without having to switch the entire editor mode. Individual HTML blocks can also be previewed in place and immediately.
>
> From Gutenberg, or the Ship of Theseus

When working with Markdwon we can make two different assumptions: Markdown is used exclusively for writing content or Markdown is used to write an entire page (both content and structure). I've always advocated the first approach. If we have to add structure to the content of a post or page I've grown used to adding HTML directly to the page rather than use a plugin or shortcode to provide the same functionality.

If we need to support new ways to add content using Markdown it's fairly easy to work at the parser level and create extensions to Markdown. The ones that come to mind the most are tables and fenced code blocks but others are possible too.

I've never been a fan of shortcodes. I'd rather take the embed code and add it directly to the page, maybe surrounded by a `div` that will help me style the content or maybe do something else related to accessibility rather than let a shortcode dictate how that works. That said I want to retain the option of doing it

my way as it may change for individual instances and I don't want to customize a block to do something that may or may not change.

The future of Child Themes

Another thing I'm not clear is what will happen with child themes and how will the concept of theme change with blocks, particularly for older content (as we'll discuss in a later section).

TO reduce the workload for users of a child theme we use a function similar to the one below to load the parent's stylesheet before we load the child theme's. This way we get all the styles from the parent theme and can make changes to the child without affecting the parent.

```
function my_enqueue_styles() {

    /* If using a child theme, auto-load the parent theme style. */
    if ( is_child_theme() ) {
        wp_enqueue_style( 'parent-style', trailingslashit( get_template_d
    }

    /* Always load active theme's style.css. */
    wp_enqueue_style( 'style', get_stylesheet_uri() );
}
add_action( 'wp_enqueue_scripts', 'my_enqueue_styles' );
```

Since themes will restrict what blocks you make available to your content creators, is something like this still necessary? How do global styles work and how do children elements and blocks inherit from the parent theme (assuming there is one)?

For years Wordpress has stated that working with Child Themes and parent frameworks is the way forward. How has this changed and what will Gutenberg's impact be in the Wordpress theme industry? Part of me hopes it's a good impact in reducing the number of developers creating garbage themes and driving prices down for the rest of the development community. But the other part hopes that not many good developers will drop oout because of the new development requirements.

How will this impact existing theme frameworks like Thesis and Genesis?

How do blocks help craft an intuitive interface?

One of the things that fly on the face of logic is that blocks will help craft intuitive interfaces. I don't think that's the case, at least not as they are currently built. They each require a selection and you have to accept the values and names of the people who created the blocks rather than taking the flexibility of being able to craft your own content in a way that is comfortable to you.

For me personally the current iteration of Gutenberg is far from intuitive. Searching for blocks and trying to use the new block search feature make authoring content much slower than traditional working flows, particularly considering that I write content outside of Wordpress and just paste it into the editor to validate and publish.

Right now there is no keyboard shortcuts that will make navigation more intuitive and less of a hassle. **This is also an accessibility issue**.

How do we work with older content?

To validate my assumptions I cloned my technical blog (The Publishing Project) and installed the Gutenberg plugin. My blog is not too sophisticated, it uses the two PHP functions below to customize behavior and CSS to add or modify the way things look on the blog.

The first function works around a bug in some versions of Chrome where Prism, my syntax highlighter, stopped working when using the clipboard add-on (that allows for one click copy of the highlighted content). It enqueues Prism and clipboard.js and the prism.css style sheet.

```
// Function to add prism.css and prism.js to the site
function add_prism() {
    wp_register_style('prismCSS', // handle name for the style so we can r
    get_stylesheet_directory_uri() . '/prism/css/prism.css' // location of
  );
  // Register clipboard.js file
  wp_register_script(
    'clipboardJS', // handle name for the script so we can register, de-re
```

```
    get_stylesheet_directory_uri() . '/prism/js/clipboard.min.js' // locat
  );
  // Register prism.js file
  wp_register_script(
    'prismJS', // handle name for the script so we can register, de-regist
    get_stylesheet_directory_uri() . '/prism/js/prism.js' // location of t
  );

  // Enqueue the registered style and script files
  wp_enqueue_style( 'prismCSS' );
  wp_enqueue_script( 'clipboardJS' );
  wp_enqueue_script( 'prismJS' );
}

add_action('wp_enqueue_scripts', 'add_prism');
```

This is what is driving me nuts. I'm trying to figure out how to do something like `add_prism` in a Gutenberg environment. Do I need to add the scripts and stylesheets every time that I ennqueue the scripts for the block? or do I only need to do it once?

I was able to install and use the Prism for WP and get it to work by eliminating the ability to do a one-button copy for each of the highlighted code blocks. BUt thew question still remains... how do I move the code from `functions.php` to a block? Is it necessary?

This is also part of my trying to understand React and Wordpress abstraction on top of it. How do you add classes beyond name or classes that are part a fixed string and part dynamic based on a value in the editor?

The second function changes the way Wordpress insert images by using a figure elemement, removing the https/http in the URL and replacing it with a protocol relative URL (using `//`) and adding a caption to the figure if there is one present

```
function html5_insert_image($html, $id, $caption, $title, $align, $url, $s
  $src  = wp_get_attachment_image_src( $id, $size, false );
  $url = str_replace(array('http://','https://'), '//', $src[0]);
```

```
  $html = get_image_tag($id, '', $title, $align, $size);
  $html5 = "<figure>";
  $html5 .= "<img src='$url' alt='$alt' class='size-$size' />";
  if ($caption) {
    $html5 .= "<figcaption class='wp-caption-text'>$caption</figcaption>";
  }
  $html5 .= "</figure>";
  return $html5;
}
add_filter( 'image_send_to_editor', 'html5_insert_image', 10, 9 );
```

But neither of them work. I can't tell if it's the migration to a new host, the Gutenberg plugin, any other existing plugin or something else. Some of the things that I've observed:

The `html5_insert_image` function is madde redundant with the image block. It doesn't matter that you've customized the way the image works, either you use the block and get the image the way Wordpress wants you to use it or you create a custom block and embed both the desired HTML result and the CSS styles in your custom element.

Is HTML that terrible?

> Content in WordPress is stored as HTML-like text in post_content. HTML is a robust document markup format and has been used to describe content as simple as unformatted paragraphs of text and as complex as entire application interfaces. Understanding HTML is not trivial; a significant number of existing documents and tools deal with technically invalid or ambiguous code. This code, even when valid, can be incredibly tricky and complicated to parse – and to understand.
>
> From The Language of Gutenberg

This is, perhaps, my biggest issue with Gutenberg and its associated views of development and content creation: **How does HTML factor in the equation?**

Understanding HTML should be the first requirement for people working on

creating front end interfaces or authoring content for use on the web. That should be non-negotiable, regardless of how trivial or non trivial it appears to be.

Wordpress doesn't carry the baggage of having to support HTML 1.0 from the early 1990s like browsers do. It's true that browsers and, to a lesser degree, authoring tools have to deal with tag soup markup to provide backwards compatibility with documents that did not enforce good authoring practices.

> "Tag soup" encompasses many common authoring mistakes, such as malformed HTML tags, improperly nested HTML elements, and unescaped character entities (especially ampersands (&) and less-than signs (<)).
>
> From Wikipedia's Tag Soup Entry

The fact that browsers have to take tag soup markup doesn't mean that Wordpress should do so as well. Wordpress HTML parsers should not accept malformed HTML and it should complain loudly when you feed it crap, although modern editors and Markdown parsers should not create bad HTML at all anymore.

So I don't really see the need to drop HTML as an authoring language and replace it with a visual editor. The reasoning is not only flawed but also dangerous and restrictive for the people who are comfortable weaving HTML with Markdown to create richer content.

And, if you really need to generate HTML programatically you can use a templating engine like the following:

- Handlebars
- Nunjucks
- Mustache
- doT
- Dust
- EJS
- Pug (formerly Jadde)

Another option is to work with Web Components, a set of open specifications to build reusable components with open standards rather than proprietary tool that was later open sourced. To me this is no different than WebKit and Chromium... yes, they are open source but it's also true that Apple and Google decide what

happens with the project as a whole.

As with many other situations, **a one size solution does not fit all cases**.

Building a Gutenberg block

Because I don't think building blocks is the easiest way to create authoring experiences for Wordpress the best way to prove if this is the case, or not, I will work on creating a simple text block that will take Markdown as its input and produce well formed HTML.

I'm particularly interested in the following questions

- Will the cutom block be able to handle long text with embeded video and fenced code blocks (supported by Jetpack's current Markdown parser)? If not, what's the alternative for older content written in that format?
- How do we apply page-wide scripts and styles (can I use my two functions outlined earlier in a Gutenberg-based system)? Can we?
- How do we ensure that older content created before Gutenberg still works in the new editor?

Why are React, JSX and Babel required?

> At the risk of igniting debate surrounding any single "best" front-end framework, the choice to use any tool should be motivated specifically to serve the requirements of the system. In modeling the concept of a block, we observe the following technical requirements:
>
> - An understanding of a block in terms of its underlying values (in the random image example, a category)
> - A means to describe the UI of a block given these values
>
> At its most basic, React provides a simple input / output mechanism. Given a set of inputs ("props"), a developer describes the output to be shown on the page. This is most elegantly observed in its function components. React serves the role of reconciling the desired output with the current state of the page.

> The offerings of any framework necessarily become more complex as these requirements increase; many front-end frameworks prescribe ideas around page routing, retrieving and updating data, and managing layout. React is not immune to this, but the introduced complexity is rarely caused by React itself, but instead managing an arrangement of supporting tools. By moving these concerns out of sight to the internals of the system (WordPress core code), we can minimize the responsibilities of plugin authors to a small, clear set of touch points.
>
> From: Gutenberg Element

For a while Wordpress Core has been a React/JSX application and has been steering all projects that touch the core of WordPress in that direction. It wasn't a problem when working themes, plugins and short codes added content to the TinyMCE editor or produced markup that was inserted directly into the editor. That meant that, as a good developer, you had to know PHP, HTML, CSS and JavaScript... The basic tools for most front-end development.

But now, because the underlying structure of the editor lives in core, Wordpress developers must learn Wordpress' version of the React ecosystem (React, Redux and the custom Wordpress layers over React), at least until something better comes up and we have to rip out all the React plumbing and start over from scratch once again.

One big question is **who chose React and its ecosystem for Wordpress and where was that communicated (if it was) to people outside the core Wordpress team?**

Onto building the block

To create a plugin with Gutenberg blocks we need at least the following files:

- An `index.php` at the root of the plugin directory that we'll use to register the blocks in subdirectories
- Inside each subdirectory containing a block
    - `.babelrc` Babel configuration files
    - `.gitignore` to choose what files to ignore when pushing to Git
    - `block.js` Block configuration
    - `editor.css`

- `index.php`
- `package.json`
- `style.css`
- `webpack.config.js`

`index.php` will point to the `index.php` file in each directory that will, in turn load the assets for each block we include. This is how we can incorporate more than one block per plugin.

```
<?php
/**
 * Plugin Name: Gutenberg Examples
 * Plugin URI: https://github.com/WordPress/gutenberg-examples
 * Description: This is a plugin demonstrating how to
 * register new blocks for the Gutenberg editor.
 * Version: 0.1.0
 * Author: the Gutenberg Team
 *
 * @package gutenberg-examples
 */
defined( 'ABSPATH' ) || exit;
include '03-editable-esnext/index.php';
include '04-controls-esnext/index.php';
include '05-recipe-card-esnext/index.php';
```

For the rest of the examples, I will use code from the Gutenberg Examples Github repository, specifically the `03-editable-esnext` block.

The first file is the `package.json` file to install babel and webpack dependencies. It also installs `cross-env` to use a single command across platforms, accounting for Windows differences to how Posix systems (OS X and Linux) work.

```
{
  "name": "03-editab"03-editable-esnext""version": "1.0.0",
  "license"-2.0-or-later",
  "main": "blo": "s",
  "devDependencies": {
```

```
    "b"devDependencies": {
    "babel-core"der": "^7.1.1",
 ": ""babel-loader": "^7.1.1",
    "babel-plugin-transform-react-jsx"env": "^1.6.0",
 ": ""babel-preset-env": "^1.6.0",
    "cross-env": "^5.0.1",
    "webpack""build": "cross-en": ""scripts": {
    "build"=production webpack",
    "dev": "cross-env BABEL_ENV=default web": ""dev": "cross-env BABEL_ENV
  }
}
```

Next we use `.babelrc` to configure Babel's behavior. We do two things:

- Configure `babel-preset-env` to go back 2 versions for most desktop and iOS browsers, last version for Android and Chrome for Android, plus IE11
- Use `wp.element.createElement` to create elements instead of the default `React.createElement`. The `wp.element.createElement` function is part of Wordpress element abstractio layer that sits on top of React/JSX.

```
{
  "presets": [
    [
      "env",
      "env"     "modules": false,
        "targets": {
          "browsers": [
            "last 2 Chrome versions",
            "last 2 Firefox versions",
            "last 2 Safari versions",
        "last 2 Chrome versions"",
            "last 1 Android version",
            "last 1 ChromeAndroid version",
            "ie 11"
          ]
        }
      }
```

```
    ]
  ],
  "last 1 Android version""plugins"rm-react-jsx",
      {
        "pragma": "wp.element.cr": ""pragma": "wp.element.createElement"
      }
    ]
  ]
}
```

After configuring Babel we need to configure Webpack. As far as configurations go, it's a simple one. We specify a single entry point, an output path, and a single module loader for JS and JSX files... We also ignore `node_modules` :D.

```
module.exports = {
  entry: './block.js',
  output: {
    path: __dirname,
    filename: 'block.build.js'
  },
  module: {
    loaders: [
      {
        test: /.js$/,
        loader: 'babel-loader',
        exclude: /node_modules/
      }
    ]
  }
};
```

The other big file in a modules is `index.php`. It uses `wp_enqueue_script()` and `add_action` to enqueue scripts and `wp_enqueue_style()` and `add_action` to load stylesheets.

The reason why we do it this way it to make sure that dependencies for both scripts and stylesheets are handled properly.

```
<?php

defined( 'ABSPATH' ) || exit;

add_action( 'enqueue_block_editor_assets', 'gutenberg_examples_03_esnext_e

function gutenberg_examples_03_esnext_enqueue_block_editor_assets() {
  wp_enqueue_script(
    'gutenberg-examples-03_esnext',
    plugins_url( 'block.build.js', __FILE__ ),
    array( 'wp-blocks', 'wp-i18n', 'wp-element' ),
    filemtime( plugin_dir_path( __FILE__ ) . 'block.build.js' )
  );

  wp_enqueue_style(
    'gutenberg-examples-03_esnext-editor',
    plugins_url( 'editor.css', __FILE__ ),
    array( 'wp-edit-blocks' ),
    filemtime( plugin_dir_path( __FILE__ ) . 'editor.css' )
  );
}

add_action( 'enqueue_block_assets', 'gutenberg_examples_03_esnext_enqueue_

function gutenberg_examples_03_esnext_enqueue_block_assets() {
  wp_enqueue_style(
    'gutenberg-examples-03_esnext',
    plugins_url( 'style.css', __FILE__ ),
    array( 'wp-blocks' ),
    filemtime( plugin_dir_path( __FILE__ ) . 'style.css' )
  );
}
```

The final file we'll discuss is `block.js`. The core of the block and what tells Gutenberg how the block works, what to show in the editor and how to style the component.

The problem I have with this is that it makes it hard to reason and figure out how to make the code work. This is not helped by the fact that different tutorials

and code examples use different (and contradictory) methods to create and load Gutenberg blocks.

The example below uses (almost complete) ES6 syntax. It still uses `const` instead of `import`. This is a Babel issue; Babel doesn't support `import` out of the box; if we want to use import in the code we'll transpile with babel we need to install the babel-plugin-syntax-dynamic-import plugin and add it our Babel configuration.

```
const { __ } = wp.i18n;
const { registerBlockType, Editable, source: { children } } = wp.blocks;

registerBlockType('gutenberg-examples/example-03-editable-esnext', {
  title: __('Example: Editable (esnext)'),
  icon: 'universal-access-alt',
  category: 'layout',
  attributes: {
    content: {
      type: 'array',
      source: 'children',
      selector: 'p'
    }
  },
  edit: props => {
    const { attributes: { content }, focus, className, setFocus } = props
    const onChangeContent = newContent => {
      props.setAttributes({ content: newContent });
    };
    return (
      <Editable
        className={className}
        onChange={onChangeContent}
        value={content}
        focus={focus}
        onFocus={setFocus}
      />
    );
  },
  save: props => <p>{props.attributes.content}</p>
```

```
});
```

The CSS has two sides, one that will work for the editor (`editor.css`) and one that will work on the frontend, what the user will see (`style.css`) The names for the classes appear to be automatically generated based on the name of the block and other details.

```
/**
 * Note that these styles are loaded *after* common styles, so that
 * editor-specific styles using the same selectors will take precedence.
 */
/* editor styles: editor.css*/
.wp-block-gutenberg-examples-example-03-editable-esnext {
  color: green;
  background: #cfc;
  border: 2px solid #9c9;
  padding: 20px;
}

/**
 * Note that these styles are loaded *before* editor styles, so that
 * editor-specific styles using the same selectors will take precedence.
 */
/* front end styles: style.css*/
.wp-block-gutenberg-examples-example-03-editable-esnext {
  color: darkred;
  background: #fcc;
  border: 2px solid #c99;
  padding: 20px;
}
```

The fact that the names for the classes is automatically generated can be problematic if you need a specific class for your code to work. For example: Prism Highlight needs a class value of `language-` plus the name of the language to highlight (`language-css` or `language-javascript`). How do you generate those class names for Gutenberg? Will that be something that I have to add by hand? Do I add it to the comment that Gutenberg generates or do I need to generate classic blocks and then add the special content there?

Licensing the code

There is also a licensing question: How is Matt Mullenweg allowing this non-GPL code into Wordpress? He has been very vigorous in the past in going after people who sold themes that used split licenses (thesis, anyone?) So what changed now?

For the longest time Wordpress (at Matt Mullenweg's insistence) has not allowed dual licensed code to be shipped with Wordpress at least for the official Marketplaces at wordpress.org/themes and wordpress.org/plugins. But React, at least since version 16, is released under the MIT License which is much more permissive license than GPL yet fully compatible with it.

So what happens to the dual licensing of code outside core? Is it OK to break the dual licensing rule now that Wordpress Core does the same thing? If it's not then what's the rationale for this disparity? Are the concerns in There is No Such Thing as a Split License still valid or do they change now that Wordpress itself has gone Dutch?

For more information about dual licensing and the issues I have with Wordpress enforcement of the GPL, see my blog post: GPL and Wordpress: Developer Beware.

Issues I'm experiencing

Apparently the full download of the export file was causing issues with the Wordpress Importer. It worked by restoring the posts first **without uploading media** and then creating a second export of only media assets and uploading it. It does not give me a confirmation (just blank screen) but I can see the media uploaded into the new installation.

I've tried migrating my blog where the oldest post is 6.5 years old. I chose to use this blog rather my personal one where the oldest post is 12 years old because I want to validate the migration with a large number of posts but not large enough that it'll make problems hard to troubleshoot.

There are times when the update process for published posts will not work. It may be related to Issue 3948 and it's **definitely related** to Issue 2565 as I am using Cloudflare as a CDN and it's challenging the post request to the Wordpress REST API as a JSON injection attack; when the API cannot respond then the request is blocked.

This is what causing posts not to update, publish or save; whish there was better error messages at least for developers and administrators.

> I have a ticket open with Cloudflare to see if it's possible to solve the issue. If it isn't then I have to decide if performance is more important than working site. In the meantime I've configured a Cloudflare page rule to trigger when it hits the wp-json endpoints while I wait for a reply from Cloudflare Support.
>
> The current workaround to add the IP addresses of the hosts where I'm working to my Cloudflare whitelist.

Related to this is a baffling error: The first time you go to a post and preview it in Gutenberg, it'll display the content. If you make any changes to content and it can't be updated you can no longer preview it or it'll hang generating the preview. The content that I want to preview is still in the Wordpress database, why do we have to validate the new changes (if any) to display the content? While this is related to the Cloudflare issue, it's bad usability and UX.

If I have the following code as the first paragraph in a block:

```
<div class="message info">
<p>paragraph content.</p>
</div>
```

Gutenberg will not close the div tag and consider the following paragraph to also be a div with `class="message info"` whether that's what I wrote or not. The solution is to change the block to classic whenever using message type CSS classes.

The content looks different when previewing it than when it's published. The post is not respecting paragraph spacing or the additional CSS I added to the theme to attempt to compensate.

I'm having to redo a lot of changes over and over when revisiting a post I've edited with Gutenberg. If I edit a blockquote block,change the HTML code and save it then, the next time I visit the post I will be promppted to overwrite, edit as classic block or edit HTML. If I overwrite the change to continue working in Gutenberg some, if not all the code, will disappear.

Alternatives

When I first heard about Gutenberg and the future of the Wordpress editor I was in the middle of experimenting with Hugo as way to build a static replacement for my labs website.

As part of that experiment I moved the content from the Wordpress blog to Hugo and was pleasantly surprised with the result. To get the content in a format appropriate for Hugo I used the Wordpress to Hugo exporter plugin for Wordpress; it creates drop in content for Hugo consisting of YAML front matter and Markdown body.

This would allow me to integrate other technologies like Service Workers, Cloudinary for responsive images and Cloudflare without having to worry if it'll break my authoring experience and it will also give me much tighter control over the final result without having to worry about trying to fit what I want to do with the way the CMS wants them done.

Conclusion: A step forward or two steps back?

In his post/tutorial Introducing Gutenberg Boilerplate For Third Party Custom Blocks!, Ahmad Awais list what he sees the pros and cons of Gutenberg. I've quoted the aspects I consideer most important from his pro/con evaluation below along with my comments about it.

How hard are we making it for new developers to create themes and plugins for Wordpress? Part of me is happy that this will reduce the amount of overtly

cheap, it may turn developers away from Wordpress altogether.

> It's way too hard for a beginner developer to understand let alone write this kind of code.

I don't contribute to core so I wouldn't know how any people are actively contributing to development of core in general or Gutenberg in particular but statements like the one below worry me enourmously.

> The fact is, there aren't many who know the ropes around here. Let alone interested in contributing to this project. Which leaves us with scary conclusions?

How large is the Wordpress team working on Gutenberg? Is the number of contributors in Github a fair aproximation of the size of the Gutenberg team?

Furthermore how did Wordpress decide to work with React (licensing issues aside) and the plugins that they chose to work with?

How do they manage the dependency tree and outdated dependencies? This is particularly troublesome because they are using Babel 7 beta, how will this affect the work people do in Gutenberg as Babel moves through beta versions into final release?

> When using NPM Packages, who and how do we decide which package should be utilized?

I see Gutenberg as the culmination of a trend I saw starting with with Calypso, the new Wordpress.com, built as the new front end for wordpress.com and desktop clients for Windows, Mac and Linux. Using a React-based application when building a multiplatform his mean tool makes some kind of sense, in my opinion. It makes much less sense to use this stack when creating the content editor framework unless you can guarantee ease of use or a shallow and fast learning curve.

So, from my perspective as a developer and a user, **what does this mean for Wordpress moving forward?**

The majority of the code remains in PHP there are portions of the codebase now being written in React and Javascript. How much of the code will transition away from PHP to React/Redux/Abstraction Layer that we've seen in Calypso and

Gutenberg?

I believe that moving portions of Wordpress to React is a step back as it reduces the number of developers that can work and contribute to the codebase and it raises the barrier of entry for new developers higher than it needs to be. This used to be less of an issue because themes and plugins only needed Javascript for additional functionality but it has become a real issue now that the equivalent of themes and plugins content are written in React with the Wordpress abstraction layer on top of it.

I also have an issue with turning everything into a block. One thing that Gutenberg hasn't addressed yet is inline content like inline code (using the `code` element or the ` character in Markdown). Right now that is not possible and I wonder what other inline elements are missing from the toolbar and from the current roadmapt for the project.

I think that Gutenberg can be an aawesoe experience if handled properly.

I'll eat React and learn it because that's what Gutenberg wants. That said I'm not holding my breath on a smooth transition with the current level of commuication.

My next task will be to try and build a block using Prism.js to replace my `functions.php` theme customization. This will be an important consiideration on whether I stay with Gutenberg or not.

Links and Resources

- General
    - Gutenberg, or the Ship of Theseus
    - Editor Technical Overview
    - Design Principles
    - With Gutenberg, what happens to my Custom Fields?
    - Is Gutenberg built on top of TinyMCE?
    - Wordpress Child Themes
    - WP-Tonic 245: Does the Genesis Framework Have a Future in a World of Theme & Page Builders?
- Opinions and explanations
    - How Existing Content Will Be Affected by Gutenberg WordPress Editor

        - Matt Mullenweg Addresses Concerns About Gutenberg, Confirms New Editor to Ship with WordPress 5.0
        - Gutenberg Editor Review: Please Don't Include This in WordPress Core
        - Thoughts on Gutenberg
        - Diving Into the New Gutenberg WordPress Editor (Pros and Cons)
- Tutorials
    - Introducing Gutenberg Boilerplate For Third Party Custom Blocks!
    - Create Guten Block Toolkit: Launch, Introduction, Philosophy, & More!
- Video Presentations
    - Matt Mullenweg: State of the Word 2017
    - Morten Rand-Hendriksen: Gutenberg and the WordPress of Tomorrow
- Wordpress CLI
    - Wordpress Scaffold and Gutenberg
    - Running WP-CLI commands remotely
    - WP-CLI commands cookbook
    - WP-CLI external resources