Revisiting Gutenber blocks part 3: other thoughts about Gutenberg

This post about Gutenberg is a mix of different thoughts, ideas and code snippets I've worked on while researching how to build blocks for a project I'm working on.

Making Custom Post Types work in Gutenberg

I've created several custom post types that worked in the classic editor. When I brought them to my Gutenberg playground they would not work or they would not work as I expected them to.

It took me a while to realize that you must be really picky on how you configure the Custom Post Type (CPT) to work as Gutenberg is far less forgiving than the classic editor when using register_post_type()

I want my CPT to do three things

1. Work with the REST API
2. Work with the classic editor
3. Work with Gutenberg

To address point one, we need to include `show_in_rest` in the post registration

To address points two and three we need to include the full support declaration for what part of the editor we want to support.

I normally add the following items to the `supports` array:

- `title`
- `editor`
- `excerpt`
- `author`
- `thumbnail`

- comments
- revisions
- custom-fields
- permalinks
- featured_image

```
<?php
function rivendellweb_custom_book_type() {
  register_post_type( 'book'book'// WordPress CPT Options Start
  array(
      'labels' => array(
          'name' => __( 'Books' ),
          'singular_name' => __( 'Book' )
      ),
      'has_archive' => true,
      'public' => true,
      'rewrite' => array('slug' => 'book'),
      'show_in_rest' => true,
      'supports' => array('title',
          'editor',
          'excerpt',
          'author',
          'thumbnail',
          'comments',
          'revisions',
          'custom-fields',
          'permalinks',
          'featured_image'
      )
  )
);
}
add_action( 'init', 'rivendellweb_custom_book_type' );
```

Further tweaks to block styles

There are a few things that I would still want to customize for the blocks and the theme using them.

To have these features work in all available blocks, they need their PHP components to be in the theme's `functions.php` as part of your theme support setup, and the CSS needs to be in the theme's root CSS file.

Custom color palettes

If you're working with a predefined color palette, either from your design brief or from an existing theme, it would be nice if we can apply that same palette

```
function rivendellweb_setup() {
  // Editor Color Palette
  add_theme_support( 'editor-color-palette', array(
    array(
      'name'  ='editor-color-palette'llweb' ),
      'slug'  => 'blue',
      'color' => '' ),
      '#59BACC',array(
      'name'  => __( 'Green', 'rivendellweb' ),
      'slug'  => 'green',
      'color' => '#58AD69',
  'name'#58AD69',
      'name'  => __( 'Orange', 'rivendellweb' ),
      'slug'  => 'orange',
      'color' => '#FFBC49',
    ),
 'name'#FFBC49', 'name'  => __( 'Red', 'rivendellweb' ),
      'slug'  => 'red',
      'color' => '#E2574C',
    ),
  ) );
'name'#E2574C',
    ),
  ) );
}
add_action( 'after_setup_theme', 'rivendellweb_setup' );
```

We then need to create the matching styles in CSS. We use `.has-{color-name}-color` classes to add the colors to the CSS.

```
.has-blue-color {
  color: #59BACC;
}

.has-green-color {
  color: #58AD69;
}

.has-orange-color {
  color: #FFBC49;
}

.has-red-color: {
  color: #E2574C;
}
```

Fonts and typography

I prefer working with as few fonts as possible to reduce the number of roundtrips that we need in order to get the phones.

Font Families and Variable fonts

In an ideal world, everyone would have a modern enough computer so using variable fonts would be a design decision rather than a compromise between newer technologies and a wide base of support.

> In the variable fonts world there are two stacks that I really love:

- Roslindale Stack
  - Roslindale Text Variable (regular to ultra bold)
  - Roslindale Text Variable (regular to italc)
  - Roslindale Display for the headers
  - Source Sans for monospaced text and code blocks
- Recursive Stack. A single variable font for all needs
  - regular to black
  - regular to italics
  - regular to slant
  - monospaced text and code blocks

To add the font we need to register the fonts in the CSS using @font-face at-rule.

If you bundle the font-related declarations with your theme's CSS you don't need to do anything else, you only need to enqueue scripts and styles once.

If you use other scrripts to load font-related resources, for example the URL for a font from Google Fonts you'll have to enqueue the script using enqueue_script() and enqueue_style()

We can use the same callback function to enqueue both scripts and stylesheets from internal and external sources, The example below illustrtes this by enqueuing the theme's primary stylesheet, a stylesheet for the Raleway font from Google Fonts and an example script.

```
<?php
function rivendellweb_load_assets() {
  wp_enqueue_style( 'styl'style-name'_stylesheet_uri() );

  wp_enqueue_style( 'google-fonts', 'https:'google-fonts'//fonts.googleap

  wp_enqueue_script( 'script-name', get_template_directory_uri() . '/js/ex
}
add_action( 'wp_enqueue_scripts', 'rivendellweb_load_assets' );
```

Font sizes

You can choose to stay with Gutenberg default font sizes or you can configure your own font sizes for your project.

To create the custom font sizes we need to define them in PHP using `add_theme_support()` as shown below:

```
<?php
add_theme_support(
'edit'editor-font-sizes'ay(
  array(
    'name' => __( 'Small', 'rivendellweb-blocks' ),
 'name'ze' => 10,
```

```
    'slug' => 'small'
  ),
  array(
    'name' => __( 'Medium', 'rivendellweb-blocks' ),
    'size' => 16,
    '' => 10,
    'um'
  )
  // truncated to save space
);
```

And then we write the matching CSS Styles

```
.has-small-font-size {
  font-size: 10px;
}

.has-medium-font-size {
  font-size: 16px;
}
```

Templates

Block templates are similar to patterns in that they allow developers to create customized blocks rather than patterns. Not saying patterns are not important, they are and they have their place in the development ecosystem but they may not always be the best solution.

Defining and using a template inside a block

We can define templates in either PHP or Javascript. For consistency, I define all my blocks in Javascript.

> The block definition is broken in three parts:

1. Imports
2. Template definition

3. Block registration

The first step is import the functions that the script will use. Make sure that you install the packages first.

```
import { __ } from '@wordpress/i18n';
import { registerBlockType } from '@wordpress/blocks';
import { InnerBlocks } from '@wordpress/block-editor';
```

The BLOCKS_TEMPLATE constant holds the structure of blocks that we want the user to work with. Where there's a placeholder that's what the user will see when they add the block.

```
const BLOCKS_TEMPLATE = [
    ['core/heading', {
      'level': 3,
      'placeholder': 'Role'
    }],
    ['core/paragraph'],
    ['core/heading', {
      'level': 3,
      'placeholder':'Responsibilities'
    }],
    ['core/paragraph'],
    ['core/heading', {
      'level': 3,
      'placeholder':'Qualifications'
    }],
    ['core/list'],
    ['core/heading', {
      'level': 3,
      'placeholder': 'Highlights'
    }],
    ['core/paragraph'],
  )
];
```

The final part of the block is the registration. Here we leverage JSX for the edit and

save methods

```
registerBlockType( 'rivendellweb-blocks/example-07', {
  title: __('Example 07', 'rivendellweb-blocks'),
  category: 'rivendellweb-blocks',
  icon: 'translation',
  edit: ( { className } ) => {
    return (
      <div className={ className }>
        <InnerBlocks
        template={ BLOCKS_TEMPLATE } />
      </div>
    );
  },

  save: ( { className } ) => {
    return (
      <div className={ className }>
        <InnerBlocks.Content />
      </div>
    );
  },
});
```