Install environment

* Local by **flywheel** // gives an address on local => start site => view site
  + Files => Click right on the name of the site // open the folder
* site/wp-admin // admin
* WordPress is not optimized for creating 10000 and 100000 posts at once, automatically

Websites and extensions

* <https://codex.wordpress.org/>
* [https://www.Developer.wordpress.org](https://developer.wordpress.org/)
* **PHP Awesome Snippets** // extension for vs code

WordPress plugins

* **regenerate thumbnails** // to help with creating new image sizes for already uploaded images
* **Advanced custom field (ACF)**
* **Advanced custom field pro** // the repeater field, the flexible content field
* **Custom Metaboxes 2 (CMB2)** // custom fields
* **Manual image crop**

React tools and libraries

* **Frontity** // build all the websites with jsx
* **@wordpress/scripts**
  + **@wordpress/data**
  + **@wordpress/api-fetch**

PHP

* array\_rand(array, how many)
* trim(string) // trim — Strip whitespace (or other characters) from the beginning and end of a string
* var\_dump($pets)
* print\_r()
* array\_filter(array, function)
* array\_map(function, array)
* number\_format()
* empty()

General Functions

* rules of thumb => the\_title() echos the value while get\_the\_title() returns the value. You need to echo it yourself.
* **bloginfo('name');** // echos the name of the site
  + **blog info('description');**
  + **bloginfo('charset');** // <meta charset="<?php bloginfo("charset") ?> >
* **get\_theme\_file\_uri('/images/apples.jpg')**
* **wp\_enqueue\_script('custom-font', get\_theme\_file\_uri( '/src/index.js'), dependencies, version, close before body tag (synch) true or false)**
* **site\_url('/about-us')** // gives the root url and the argument is added to it. Local by flywheel gives a real developer domain while xampp and wampp don’t. so it's better we don't use /about-us in the link.
* **get\_the\_ID()** // id of current page
* **wp\_get\_post\_parent\_id()**

wp\_get\_post\_parent\_id(get\_the\_ID()) // get the id of parent of current post

* **get\_the\_title()** // allows for id compared to the\_title(). Returns value. If gets 0 means the current page.
* **get\_permalink()** // can pass an id, and returns the value. Should be echoed
* **the\_permalink()**
* **wp\_list\_pages()** // creates a link (in li) for all the pages of the site, the argument needs an **associative array**. We associate a key to each value of the array. We use this sign => to associate the key.
* **Get\_template\_part()**
* **get\_pages()** // similar to above but does not show it it return in memory

<?php

$testArray = get\_pages(array(

'child\_of' => get\_the\_ID(),

));

if($theParent or $testArray){ // if it has a parent or has children ?>

<div class="page-links">

<h2 class="page-links\_\_title"><a href="<?php echo get\_permalink($theParent) ?>"><?php echo get\_the\_title($theParent) ?></a></h2>

<ul class="min-list">

<?php

if($theParent){

$findChildrenOf = $theParent;

}else{

$findChildrenOf = get\_the\_ID(); }

wp\_list\_pages(array(

'title\_li' => '', // don't put any spaces between names

'child\_of' => $findChildrenOf, // should be underscore

'sort\_column' => 'menu\_order', // orders can be set in the admin

));?>

</ul>

</div>

<?php } ?>

* **language\_attributes()** // put it in the html tag. Human language means English or something else. Lang="en-CA"
* **body\_class()** // in body tag. It creates some class names we can use in css and js
* **wp\_trim\_words(get\_the\_content(), 18)** // the first argument what text, the second the number of words we want to limit. Needs echo.
* **esc\_attr()** // when echoing value of input, get\_the\_title()
* **strip\_tags()**
* **esc\_functions**
  + esc\_html()
  + esc\_textarea()
  + esc\_atrr()
  + esc\_url() // when we were using site\_url()
* **sanitize\_()**
  + sanitize\_textarea\_field() // removes the html tags
  + sanitize\_text\_field // removes the html tags
* **is\_single()** // is is the single page
* **is\_main\_query()** // sometimes we have other queries for header footer … check if it is the main query
* **wp\_redirect** // use exit after redirecting
* **wp\_safe\_redirect**
* **current\_user\_can('administrator')** // only admin can

Create a New Theme:

* New folder in theme => index.php and styles.css
* In styles.css:

/\*

Theme Name: Painting Site

Author: Alma

Version: 1.0

\*/

* Creating a preview image // an image named screenshot.png 200\*900

Responsive

* <meta name="viewport" content = "width=device-width, initial-scale=1">

Posts

* **the\_ID()**
* The loop is the heart of WordPress even for one post it is used
* **have\_posts()**
* **the\_post()**

while (have\_posts()) {

the\_post(); ?>

<h2 style="color: blue"><a href="<?php the\_permalink() ?>"><?php the\_title() ?></a></h2>

<p><?php the\_content() ?></p>

<?php

}

* **the\_title()**
* **the\_content()**
* **the\_permalink()** // slug, post link. It will use **index.php** if **single.php** does not exist. But loops through only one post which we clicked on.
* The pages also use an index if we don't set a **page.php** in the theme folder
* **the\_excerpt()** // shows a little of the body.
* **the\_author\_posts\_link()**
* **the\_time('F');** // check formatting date in codex. Arguments for example can be:
  + F month
  + Y Year
  + n.j.y n-j-y n/j/y // day-month-year
  + M // abbreviation of the month
* **get\_the\_category\_list(', ');** // returns the value, which needs to be used with echo.
* **get\_post\_type()**
* **the\_post\_thumbnail()** // show the featured image(). It adds the tag image itself.
* **the\_post\_thumbnail\_url();**

post related functions

* **count\_user\_posts(user id, post type)**

Sanitize functions

* **sanitize\_textarea\_field()**

Header and Footer

* **header.php** // file names should be the same so we can use the functions below.
* get\_header()
* **footer.php**
* get\_footer()
* wp\_head(); // loads the css files but needs to be set in the function.php

functions.php

* **It is not a template file, connects to wp**
* **It runs before all files, that's why hooks are needed**
* **add\_action()**
* **add\_filter()**

add\_action(a, b), add\_filter()

* gives the wp instructions.
* a is what type of instruction. The hook we want. When we want to call the function defined in b.
* b is the function we want to run

Hooks

action

* wordpress function we want to hook on to
* **wp\_enqueue\_scripts** // for the css files
  + wp\_enqueue\_script()

<?php

function site\_resources()

{

wp\_enqueue\_style('site', get\_stylesheet\_uri()); // name of the style and the address to file

}

add\_action('wp\_enqueue\_scripts', 'site\_resources');

* **after\_setup\_theme** // to create features
  + **add\_theme\_support('title-tag')** // includes the title in the browser bar

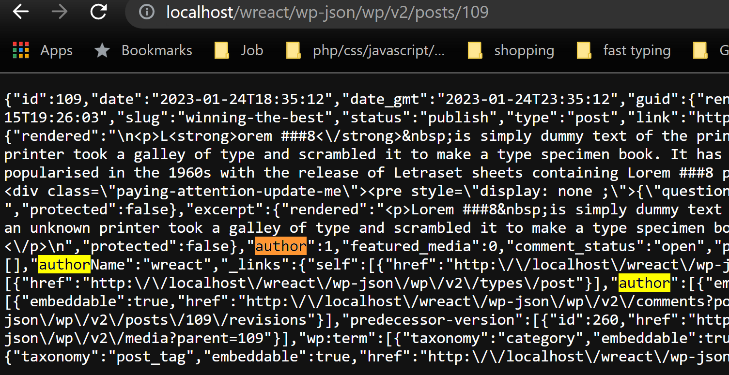
function uiversity\_features(){

add\_theme\_support('title-tag'); // what feature of the function we are interested in: title-tag

}

add\_action('after\_setup\_theme', 'university\_features');

* + **register\_nav\_menu('header-menu-location', 'Header menu location')** // second name is what shows up in admin appearance
* **init** 
  + register\_post\_type('name', array())
  + wp\_register\_style()
  + wp\_register\_script()
  + register\_block\_type()
* **pre\_get\_posts** // default queries. it passes an argument to our function we can use functions: that sends the query to the callback and we can order and refine it.
  + $query->set('order', 'ASC');
  + **set** // similar to custom query set 'post\_per\_page' for example
* **wp\_insert\_post\_data** // Used to force data to posts, very powerful
  + **add\_filter('wp\_insert\_post\_data', 'makeNotePrivate', 10, 2);** // pass two argument
  + ($data, $postarr) // $post\_arr => wp\_insert\_post\_data does not pass the id by $data - first argument, we need a second argument
* **rest\_api\_init** // action hook, add a new property to the rest api response
  + **register\_rest\_route('name','rout',array())** // ava/v2/search
  + **register\_rest\_field("post type", "filed name", array('getcallback'=>function x(){return y}))** // It will add a new field (item) to the returned api response, for example, authorName



* **admin\_menu** // to add to the admin menu list, in new plugins
  + **add\_options\_page**
* **admin\_init** // create the page section for the new plugin, it has the url name
  + add\_settings\_section // how many sections do you want
  + add\_settings\_field // What section, what url (not all the name), argument for html callback
  + register\_setting // group name for the nonce, sanitize, and default
* **after\_setup\_theme**
  + **add\_theme\_support**
  + **add\_image\_size**
  + **add\_theme\_support**
  + **add\_editor\_style**
* **wp\_loaded** // refine website based on rule, ex. Subscriber users do not see the admin menu
  + **wp\_get\_current\_user()**
* **login\_enqueue\_scripts** // styles for login
  + **wp\_enqueue\_style**
* **login\_headertitle** // set the title for the login page
  + return get\_bloginfo('name');
* **activate\_new-database-table/new-database-table.php** // runs each time the plugin gets activated, and once
* **admin\_head** // will run anytime the admin site is refreshed
* **admin\_post\_(name of hidden input in form)** //This hook helps to navigate our posted form values to our main plugin page, in which we define this. Must be logged in. The form should have this action value:

<form action="<?php **echo esc\_url(admin\_url('admin-post.php'))** ?>" class="create-pet-form" **method="POST"**>

<p>Enter a name for a new pet. Its species, weight, and other details will be randomly generate.</p>

<input type="hidden" name="action" value="createpet">

* **admin\_post\_nopriv\_(name of hidden input in form)** // for notloggedin users only

filter

* the\_content // before storing it in db
* login\_headerurl // make the link in a login page to go to our website
  + return esc\_url(site\_url('/'))
* template\_include // like the content filter but for the template of a page
* allowed\_block\_types\_all
* query\_vars

Adding a new style file, or font-awsome, or font, or js

* we have to use functions.php // don't include the **http part** for the url. Include the // after http. Hard load.

<?php

function site\_resources()

{

wp\_enqueue\_style('site', get\_stylesheet\_uri());

wp\_enqueue\_style('font-awsome', '**//**maxcdn.bootstrapcdn.com/font-awesome/4.7.0/css/font-awesome.min.css'); //name does not matter

wp\_enqueue\_style('custom-font', '**//**fonts.googleapis.com/css?family=Roboto+Condensed:300,300i,400,400i,700,700i|Roboto:100,300,400,400i,700,700i');

wp\_enqueue\_**script**('custom-font', get\_theme\_file\_uri( '/src/index.js'), null, '1.0', true);

}

add\_action('wp\_enqueue\_scripts', 'site\_resources');

* micortime() Avoid caching in development // We can change the version of the js file with. Disable cache in developer tools will disable cache for everything and all site which is not good.

<?php

wp\_enqueue\_**script**('custom-font', get\_theme\_file\_uri( '/src/index.js'), NULL, **microtime()**, true);

wp\_enqueue\_style('custom-style2', get\_theme\_file\_uri( '/css/style2.css'), NULL, **microtime()**);

}

add\_action('wp\_enqueue\_scripts', 'site\_resources');

How to connect WordPress to a theme HTML files?

* We use html in header footer and index.php
* **get\_theme\_file\_uri** // to access resources. Three folders for css, images, and src in the root folder to include css and images and javascript files.

background-image: url(<?php echo **get\_theme\_file\_uri**( '/images/library-hero.jpg' ); ?> // the root theme folder and then go on

Interior Pages

* The page.php in the root folder of theme // It is for the single normal pages like about-us
* archive-(post type name) // for archive address of post types, like/programs/, we didn't write for professors so, we don't have /professors
* single-(post type name) // for single of that post type
* page-(name of the page) // to create a different theme for a page(usually made in the admin), it does not have to be for single, we can create a search page in admin and define what we want to do with it with this template
* index.php // for blog posts address: /blog
* archive.php // for category pages, users, … if does not exist it uses index.php

Navigation Bar

* If we are creating a navigation bar for a customer client keep it in html, but if it's for a general theme put it in the admin
* Add menu location //This will add the option in admin we can choose. Without this we have the error 'Your theme does not support navigation menus or widgets.'

function uiversity\_features(){

**register\_nav\_menu('header-menu-location', 'Header menu location');**

add\_theme\_support('title-tag'); // what feature of the function we are interested in: title-tag

}

add\_action('after\_setup\_theme', 'university\_features');

* wp\_nav\_menu() // gives the list of the menu we can show. Accepts an array.

<?php wp\_nav\_menu(array(

'theme\_location' => 'header-menu-location'

)); ?>

* The hard-code way:
* is\_page('about-us') // it accepts the slug

<ul>

<li <?php if(is\_page('about-us') or wp\_get\_post\_parent\_id(0) == 30) echo 'class="current-menu-item"' ?>><a href="<?php echo site\_url('/about-us'); ?>">About Us</a></li><!-- 0 means look up the current page we can pass get\_the\_ID().

<li><a href="#">Programs</a></li>

<li><a href="#">Events</a></li>

<li><a href="#">Campuses</a></li>

<li><a href="#">Blog</a></li>

</ul>

Blogs

* Create a home page and blog page and in admin, ->setting->reading set them for the homepage and post page. It makes the blog the index we made! It is confusing. But the **index** will be used for the **blog listing screening**. And we create a file named **front-page.php** which will be used for the home page.
* Use the loop with the\_post() and do what you need!

<?php

**while (have\_posts()) {**

**the\_post();**

?>

<div class="post-item">

<h2 class="headline headline--medium headline--post-title"><a href="<?php the\_permalink() ?>"><?php the\_title() ?></a></h2>

<div class="metabox">

<p>Posted by <?php the\_author\_posts\_link() ?> on <?php the\_time('n.j.y') ?> in <?php echo get\_the\_category\_list(', ') ?></p>

</div>

<div class="generic-content">

<?php the\_excerpt() ?>

<p><a class="btn btn--blue" href="<?php the\_permalink() ?>">Continue reading &raquo;</a></p>

</div>

<!-- &raquo; right angle quote -->

</div>

<?php

**}**

?>

* **paginate\_links()** // creates pagination

Single blog post

* **site\_url('/blog')** // to go back to blog home

Category pages

* Wordpress prefers to use a theme file rather than the index for most things.
* **archive.php** //similar to the index with different titles and stuff
* **is\_category()**
* **single\_cat\_title()** //echos

<h1 class="page-banner\_\_title"><?php if (is\_category()) {

single\_cat\_title();

}

if (is\_author()) { // this can have a lot of ifs for dates and …

echo "post by: ";

the\_author();

} ?></h1>

* **the\_archive\_title()** // does the same job as writing ifs as above.
* **the\_archive\_description()** // we have to set the desc for author and category in admin

Custom Query

* Normal – default queries // wordpress based on the url automatically will query the content
* Custom // don’t rely on the URL, whenever we want what we want.
* **WP\_Query** // a **class** that accepts associative array
* **wp\_reset\_postdata(); /**/ clean up after custom query. Will reset different WordPress data and global variables back to the state it was when it was based on URL. Put it after the loop.

<?php

$homePagePosts = new WP\_Query(array(

'posts\_per\_page'=> 2,

'category\_name' => 'Awards'

));

while ($homePagePosts->have\_posts()) {

$homePagePosts->the\_post(); ?>

<div class="event-summary">

<a class="event-summary\_\_date event-summary\_\_date--beige t-center" href="<?php the\_permalink() ?>">

<span class="event-summary\_\_month"><?php the\_time('M')?></span>

<span class="event-summary\_\_day"><?php the\_time('d')?></span>

</a>

<div class="event-summary\_\_content">

<h5 class="event-summary\_\_title headline headline--tiny"><a href="<?php the\_permalink() ?>">

<?php the\_title()?>

</a></h5>

<p><?php echo wp\_trim\_words(get\_the\_content(), 18); ?><a href="<?php the\_permalink() ?>" class="nu gray">Read more</a></p>

</div>

</div>

<?php

}

wp\_reset\_postdata();

?>

* orderby
  + post\_date //default
  + title // reverse alphabetic ordering.
  + Rand
  + meta\_value // Suited for words. In WordPress, meta-data means extra or custom data associated with the post.
  + meta\_value\_num // Suited for numbers.
* posts\_per\_page
  + -1 // all posts that match
* order
  + DESC // default
  + ASC
* meta\_key
  + Post type // event for example
* meta\_query // takes an array for multiple filters.
  + Key // What we want to compare
  + compare // what kind of comparison
    - >=
  + value // with what we want to compare. For example date('ymd') is today.
  + Type // what kind the value is
    - numeric

$homePageEvents = new WP\_Query(array(

'posts\_per\_page' => -1,

'post\_type' => 'event',

'meta\_key' => 'event\_date',

'orderby' => 'meta\_value\_num',

'order' => 'ASC',

'meta\_query' => array(

array(

'key' => 'event\_date',

'compare' => '>=',

'value' => $today,

'type' => 'numeric',

) )));

* **wp\_reset\_postdata()** // resets the global post object to the default URL-based query. it is needed after each custom query and its related loop if we want to loop through another custom query that relies on **ids** or otherwise the second query will be nor work correctly. The id is changed to the first custom query but we need the page id for the second query.
  + **the\_ID()** // if we see the id for the page and query

Workflow (auto refresh and more)

* proxy preview server // don't need to refresh the page when I make changes in the php file.
* Command line
* **Node.js** // Automation is usually enabled by which lets computers run a program by javascript outside of the context of a web browser. It's a very powerful
* By node, we can use npm which lets us download tools and use them by command line.
* **Gulp** // makes automated tasks very easy
* **npm install gulp-cli -g** // install it globally
  + **gulp -version**
  + [**Error: Local modules not found in**](https://stackoverflow.com/questions/38103889/local-gulp-not-found-try-running-npm-install-gulp)

npm link gulp

* if you have **vagrant/my virtual Box**, the files are already there
  + we change the directory to the root of our project
  + it contains four files **gulpfile.js, package.json, setting.js, webpack.config.js.** //These files help to turn our computer to our robot assistant
* If you have **xampp**, docker or download the files.
  + Download the files from: [https://github.com/LearnWebCode/vagrant-lamp/tree/master/fictional-university](#_Hlk125398993" \s "1,6638,6716,179,,https://github.com/LearnWebCode/)
  + Change the values of setting.js to address the theme folder we want to track relative to where setting.js is.
* **package.json** // It's like a list of tools, packages, and scripts that we need. We use npm install and it will look into package.json to look for the packages that need to be installed.

gulp Commands

* **gulp watch**
* It opens a url that runs the website and reloads on file changes.
* **External:** [**http://10.19.0.4:3000/wreact/**](http://10.19.0.4:3000/wreact/) //This can be used on a mobile on the same network
* If the browser does not reload use **done()** after browserSync.reload() in gulp.js

CSS

* Create one css file out of different files named style.css. But we need the comment part that WordPress needs. For this, we copy that part to the css that has all files to create a bundle. WordPress only knows the style.css in the theme root, other css files are for us to organize.
* **gulp.js**

Javascript

* For javascript, we use the same method, a file that imports all needed js files and bundles them.
* For this we use **webpack.config.js**
* To use Sass you may need to install sass-loader and css-loader.
* Add this to the rule in webpack.config.js rule

{

test: /\.s[ac]ss$/i,

use: [

// Creates `style` nodes from JS strings

//'style-loader',

// Translates CSS into CommonJS

'css-loader',

// Compiles Sass to CSS

'sass-loader',

], },

* gulp scripts // in cmd also for js files
* install this module:

npm install --save-dev sass

* In gulpfile.js

const sass = require('gulp-sass')(require('sass'));

const bundleSass = () => {

return gulp.src(settings.themeLocation + 'css/style.scss')

.pipe(sass().on('error', sass.logError))

.pipe(gulp.dest(settings.themeLocation))

}

* gulp bundleSass // in the command line

Custom Post Types

* [wordpress dashicons](https://developer.wordpress.org/resource/dashicons/#update-alt) //
* register\_post\_type() //
  + **'public' => false** // don't show it in public queries or search results only, specific to the user account, it also hides in the dashboard
  + **'show\_ui' => true** // to show it dashboard

<?php

function create\_new\_post\_type(){

register\_post\_type('event', array(

'public' => true,

'labels' => array( // the names that are shown in wordpress

'name' => 'Events', // what it is shown in backend

'add\_new\_item' => 'Add New Event',

'edit\_item' => 'Edit Item',

'all\_items' => 'All Events',

'singular\_name' => 'Event'

),

'menu\_icon' => 'dashicons-calendar',

));

}

add\_action('init', 'create\_new\_post\_type');

* We shouldn’t put this in the function of a theme to access it if we switch themes. Even if we create a plugin we may deactivate it.
* **mu-plugins** folder in wp-content // Must-use plugins. Every PHP file in this folder will be automatically activated. The name does not matter.

Display

* Add a new customer query

$homePageEvents = new WP\_Query(array(

'post\_per\_page' => 2,

'post\_type' => 'event',

)); // use this variable to address the have\_posts() and the\_post()

* **single page not found issue** -> rebuild WordPress's rewrite or permalink structure. Settings -> permalink => just save without changing anything. WordPress only changes the permalink structure at key moments for performance issues.
* To fix the archive page (event list) not found issue -> we have to say our new post type supports archive. use 'has\_archive' => true when defining the new post type. if it's not working wait and restart the server or save the permalink again for the new building.

Set up a new single for the new post type

* single-event.php // a dash and the keyword of the post type.

control keywords for the slug

* 'rewrite' => array('slug'=> 'events')

function new\_post\_type()

{

register\_post\_type('event', array(

'supports' => array('title','editor','excerpt'),

'rewrite' => array('slug'=> 'events'),

'has\_archive' => true,

'public' => true,

'labels' => array( // the names that are shown in wordpress

'name' => 'Events',

'add\_new\_item' => 'Add New Event',

'edit\_item' => 'Edit Event',

'all\_items' => 'All Events',

'singular\_name' => 'Event',

),

'menu\_icon' => 'dashicons-calendar',

));

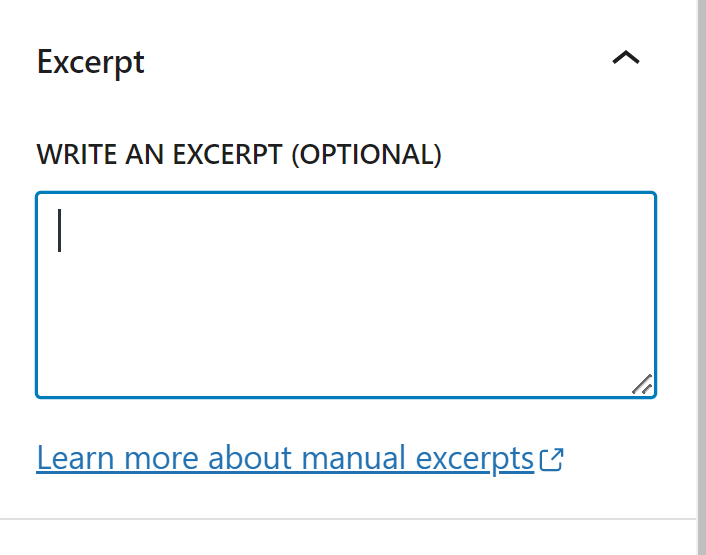
}

add\_action('init', 'new\_post\_type');

* get\_post\_type\_archive\_link('event') // To get the link for the post type, as we may change the slug again.

Handcrafted excerpt

* fill the form in the backend



* **the\_excerpt()** // For those post that does not have an excerpt takes the 50 first words, which can be changed globally but it's not good. This may create some margins when echoed internally.
* **get\_the\_excerpt()**
* **has\_excerpt()** // We can combine it with **wp\_trim\_words()**
* **custom post types** //We need to set the option when we define the custom post types.
  + 'supports' => array('title', ' editor', 'excerpt'),

Custome Fields

* **'custom-field'** // add it to the supports option of register\_post\_type()

…

register\_post\_type('event', array(

'supports' => array('title','editor','excerpt', 'custom-fields'), …

* We customize the custom field section of the admin by writing code but it's better to use a plugin if that is not necessary.
  + Advanced custom field (ACF)
  + Custom Metaboxes 2 (CMB2)

Advanced custom field (ACF)

* It does not need a custom-field option anymore.
* It revolves around field groups. But it does not need to have multiple fields. It can be used for one.
* Location rules // where it will appear. Set to the post type we want.
* **the\_field('event\_date')** // we can access this in the loop
* **DateTime()** // a class of php.to parse dates
  + **format()** // a method

<?php

$date = get\_field('event\_date', false, false); // two parameters are for not returning an error on DateTime()

if ($date) {

$eventDate = new DateTime($date);

echo $eventDate->format('M');

}

?>

</span>

<span class="event-summary\_\_day">

<?php

if ($date) {

echo $eventDate->format('d');

}

?> ?>

Manipulate default url based queries

* We can write our own query but that will take more time and we need to do some more steps for pagination. Not always the first answer.
* "If every tool you have is a hammer everything you have will look like nails. "
* This will be very general, it affects all queries front and admin.

function university\_adjust\_queries($query)

{

$query->set('posts\_per\_page', '1');

}

add\_action('pre\_get\_posts', 'university\_adjust\_queries');

* So we need to write the conditions
  + is\_admin()
  + is\_post\_type\_archive('event' and $query->is\_main\_query()))
  + is\_main\_query()

function university\_adjust\_queries($query)

{

if (!is\_admin() and is\_post\_type\_archive('event' and $query->is\_main\_query())) {

$query->set('posts\_per\_page', '1');

}

}

add\_action('pre\_get\_posts', 'university\_adjust\_queries');

Add a page for past events and pagination

* How to use a page created in admin to do something else for example show the past events. It is by default powered by page.php
* **page-(slug).php** // For this we create a file named page-(slug of the page we made). For example past-events/
  + default url based queries // The page itself is queried because it is a page we made.
* **Pagination** // paginate\_links() works with the default queries that are tied with the URL
  + **paginate\_links(array())** // set it so it can know the custom query
    - 'total' => $pastEvents->max\_num\_pages// to say how many pages we have
  + 'paged' => get\_query\_var('paged', 1) // in the custom query to say what page we are in. It can get a lot of information about the URL. '1' is the default if it can not find the variable paged from URL.

paginate\_links(array(

'total' => $pastEvents->max\_num\_pages

));

Relationship between two post types

* After creating the new post type we can go to custom fields and create a field with a relationship type.
* The relationship is an array when returned you can use print\_r to see what the variable has.
  + We need to loop through it since it is a loop.

<?php

$relatedPrograms = get\_field('related\_programs');

// print\_r($relatedPrograms); // prints all sort of information about that variable print\_r

if($relatedPrograms){

echo '<hr class="section-break">';

echo '<h2 class="headline headline--medium"> Related Program(s) </h2>';

echo '<ul class="link-list min-list">';

foreach ($relatedPrograms as $program) {

// echo $program->post\_title;

?>

<li><a href="<?php echo get\_the\_permalink($program) ?>">

<?php echo get\_the\_title($program); // accepts an id or a post object?>

</a></li>

<?php

}

echo '</ul>';

?>

<?php

}

?>

* For the other post\_type – here is the program, we can use the **custom query** to access the information.
  + Add another meta query
  + When WordPress saves the related programs for an event, we chose three (12, 120,1200)posts(programs) and related ids **will not** be stored in an array, so php first will serialize the array to something like a:3:{i:0;i:"12";i:1;i:"120";i:2;i:"1200"}. WordPress wraps each number in the quote so, to get the ids from that field, we want to look for "12" and if it contains it so we add the id to double quotes.

$today = date('Ymd');

$homePageEvents = new WP\_Query(array(

'posts\_per\_page' => 2,

'post\_type' => 'event',

'meta\_key' => 'event\_date',

'orderby' => 'meta\_value\_num',

'order' => 'ASC',

'meta\_query' => array(

array(

'key' => 'event\_date',

'compare' => '>=',

'value' => $today,

'type' => 'numeric',

),

array(

'key' => 'related\_programs',

'compare' => 'LIKE',

'value' => '"'.get\_the\_ID().'"'

)

)

));

Featured Images

* We can use content section to add images but we want to assign an image to a post (featured images or post thumbnail).
* A post can have one single featured image.
* To enable the **feature**. You need to go to funtions.php -> after\_setup\_theme action -> **add\_theme\_support(''post-thumbnails')** function.

function university\_features()

{

add\_theme\_support('post-thumbnails'); // enables only for **blog post**

}

add\_action('after\_setup\_theme', 'university\_features');

* To add the featured image to **custom post type** => in mu-plugins

//program post type

register\_post\_type('professor', array(

'supports' => array('title','editor', '**thumbnail'**),

'public' => true,

'labels' => array( // the names that are shown in wordpress

'name' => 'Professors', // what it is shown in backend

'add\_new\_item' => 'Add New Professor',

'edit\_item' => 'Edit Professor',

'all\_items' => 'All Professors',

'singular\_name' => 'Professor',

),

'menu\_icon' => 'dashicons-welcome-learn-more',

));

}

add\_action('init', 'new\_post\_type');

* **the\_post\_thumbnail('name of the added image size')** // show the featured image(). It adds the tag image itself.
* **the\_post\_thumbnail\_url();**
* **Image size** // wordpress by default saves 5 images when we upload an image with different sizes – in upload folder.
  + We can create an image size that we want and **save it**.
* **add\_image\_size('name', width, height, crop or keep the image as whole?) //**To create a new image size. You need to go to funtions.php -> after\_setup\_theme action function.

function university\_features()

{

add\_image\_size('professorLandscape', 400, 260, true); // it won't update the already uploaded images.

add\_image\_size('professorPortrait', 480, 650, true);

}

* **regenerate thumbnails** // a plugin to create the defined sizes for the prev uploaded images
* Show the added image sizes

<?php the\_post\_thumbnail('professorLandscape') ?>

**src**="<?php the\_post\_thumbnail\_**url**('professorLandscape'); ?>"

* **Cropping images**
* **Manual image crop** // a plugin that helps with the below, since we can't set something like this for all images.

add\_image\_size('professorLandscape', 400, 260, array('left','top'));

Other Images

* Some pages like posts and about us pages can use the featured image for a banner but a page like a single professor that already has used the feature image can not.
* We can set up a new field. To make it work for all the posts, set equals and not equals on the post.

function university\_features()

{

add\_image\_size('pageBanner', 500, 350, true);

}

* To show it

<div class="page-banner\_\_bg-image" style="background-image: url(

<?php $pageBanner = get\_field('page\_banner\_background\_image'); //it's an array, use get\_ not the\_!

echo $pageBanner['url']; // to access the array

?>)"></div>

* Or based on sizes

echo $pageBanner['sizes']['pageBanner'];

Reduce duplicate code

* **function** //If you want to use argument to be more flexible use this option. be careful with the\_ and get\_ functions.

function pageBanner($args = NULL){ // $args is not required

if(!$args['title']) $args['title'] = get\_the\_title();

if(!$args['subtitle']) $args['subtitle'] = get\_field('page\_banner\_subtitle');

if(!$args['photo']){

$pageBanner = get\_field('page\_banner\_background\_image');

if($pageBanner){

$args['photo'] = $pageBanner['sizes']['pageBanner'];

} else{

$args['photo'] = get\_theme\_file\_uri('/images/ocean.jpg');

}

}

* **get\_template\_part('address', 'name')** //When we have just php html that doesn't change use this option. If the second argument is given WordPress adds a dash to the first address-(second name) and tries to find a file with that criteria.

while ($homePageEvents->have\_posts()) {

$homePageEvents->the\_post();

get\_template\_part('template-parts/event'); // not the .php just the slug

}

* The second argument is helpful when for example we want to create different formatting for all search result pages for different post types and we can set content-(post name) as the name of the file so it becomes more dynamic.

while ($homePageEvents->have\_posts()) {

$homePageEvents->the\_post();

get\_template\_part('template-parts/content', get\_post\_type()); // conten-event.php

}

Add a field for Google map

* We may need a Google map **API key**. It's a handshake.
* For localhost addresses also google gives a free pass. But you need billing for that even though you don't pay.
* Otherwise, you probably are importing an iframe that linked to a webpage that lived on a google server

function universityMapKey($api){ // the $api is available to us by acf, we change it return it.

$api['key'] = 'google maps api key';

return $api;

}

add\_filter('acf/fields/google\_map/api', 'universityMapKey'); // google map is the type of the field.

* In enqueue script action:

wp\_enqueue\_script('googleMap', '//maps.googleapis.com/maps/api/js?key=google map api', null, microtime(), true);

* In front-end temp

while (have\_posts()) {

the\_post();

$mapLocation = get\_field('map\_location'); // this is for acf

?>

<!-- the class marker is for javascript to easily find the element -->

<div class='marker' data-lat='<?php echo **$mapLocation['lat']**; ?>'

data-lng='<?php echo **$mapLocation['lng']**; ?>' >

<h3><a href="<?php the\_permalink() ?>"><?php the\_title(); ?></a></h3>

<?php echo **$mapLocation['address']** ?>

</div>

<?php

}

* The Google map js is available by acf. We add to a js file.
* To **bundle** all of our js codes:

import "../css/style.scss"

// Our modules / classes

import MobileMenu from "./modules/MobileMenu"

import HeroSlider from "./modules/HeroSlider"

import GoogleMap from "./modules/GoogleMap"

// Instantiate a new object using our modules/classes

const mobileMenu = new MobileMenu()

const heroSlider = new HeroSlider()

const googleMap = new GoogleMap()

alert('hello this is a test')

* Then to create the **bundle**, go to where your gulpfile.js resides in the command line either
  + gulp watch // command to spin up the local server
  + gulp scripts

Live Search – javascript

* Creating overlay
  + Footer // after footer tag

import $ from 'jquery';

class Search {

// section 1. describe and create/initiate our object

constructor(){

this.openButton = $(".js-search-trigger");

this.closeButton = $(".search-overlay\_\_close")

this.searchOverlay = $(".search-overlay")

// let the browser know as soon as the object is born

this.events();

// this property is done for avoiding keep firing and avoiding hasClass

this.isOverlayOpen = false;

}

// 2. events => connect methods with object

// on this.head feels cold, wearsHat

// on thi.brain feels hot, goingSwimming

events(){

// On method changes this towards the html element that got clicked on, remember use bind

this.openButton.on('click', this.openOverlay.bind(this))

this.closeButton.on('click', this.closeOverlay.bind(this))

// all browser knows by s. Keyup: release the key

// when you release the key, only once

// keydown as long as you hold keep firing

// this is not a costly event but it's still bad habit.

$(document).on("keydown", this.keyPressDispatcher.bind(this))

}

// 3. methods (function, action...)

keyPressDispatcher(e){

// numerical key code check this to find e.keyCode for the key

// we can use !this.searchOverlay.hasClass('search-overlay--active') but reading from dom is slower than reading from js var

if(e.keyCode == 83 && !this.isOverlayOpen){

this.openOverlay();

}

//esc key

if(e.keyCode == 27 && this.isOverlayOpen){

this.closeOverlay();

}

}

openOverlay(){

this.searchOverlay.addClass('search-overlay--active')

// overflow hidden

$('body').addClass("body-no-scroll")

this.isOverlayOpen = true

}

closeOverlay(){

this.searchOverlay.removeClass('search-overlay--active')

$('body').removeClass('body-no-scroll')

this.isOverlayOpen = false

}

}

export default Search

Timing

* In event method // use var instead of calling from dom

this.searchField.on("keydown", this.typingLogic.bind(this))

* setTimeout(function run, min) // in constructor to reset it.

this.typingTimer;

getResults(){

this.resultsDiv.html("imagine")

this.isSpinnerVisible = false

}

Loading spinner icon

typingLogic(){

// we want to send request only after a specific time

//above the setTimeout so if a new key is hit, it is reset to 2000, 2000 gap between key strokes are needed

clearTimeout(this.typingTimer)

this.typingTimer = setTimeout(function(){

}, 2000)

}

* In event method // use var instead of calling from dom
* in footer

<div class="container">

<div class="search-overlay\_\_result"></div>

</div>

* in js
  + constructor

this.resultsDiv = $("#search-overlay\_\_result")

this.isSpinnerVisible = false;

* after clearTimeout

if(!this.isSpinnerVisible){

this.resultsDiv.html('<div class="spinner-loader"></div> ')

this.isSpinnerVisible = true

}

If the search doesn't change don’t send a request => any effect on control keys

* A variable that keeps track of search value // prev value
* key up// If you use key-down, it fires so quickly that even the search field doesn't get a chance to **update its own value**.
  + this.searchfield.on("keyup", this.typingLogic.bind(this))

Stop loader and timeout for an empty value

* Check the input value

typingLogic(){

// we want to send request only **after a specific time**

//above the setTimeout so if a new key is hit, it is reset to 2000, 2000 **gap between key strokes are needed**

// .value returns undefined should be val()

if(this.previousValue != this.searchField.val()){

clearTimeout(this.typingTimer)

if(this.searchField.val()){

if(!this.isSpinnerVisible){

this.resultsDiv.html('<div class="spinner-loader"></div> ')

this.isSpinnerVisible = true

}

this.typingTimer = setTimeout(this.getResults.bind(this), 2000)

}else{

this.resultsDiv.html('');

// since html is empty

this.isSpinnerVisible = false

}

}

this.previousValue = this.searchField.val();

}

* S key on another input

if(e.keyCode == 83 && !this.isOverlayOpen && !$("input, textarea").is(':focus')){

this.openOverlay();

}

Fetching results

* <http://localhost/wreact/wp-json/wp/v2/posts/> // wordpress rest api, use the data outside of php programming, CRUD JavaScript object notation (Json). Check developer.wordpress.org/rest-api
* Postman // a program that helps with APIs
  + ?per\_page =2
  + http://localhost/wreact/wp-json/wp/v2/posts/7 // id
  + ?search = award
* $.getJSON(url, fun(data){}) // jquery, when the getJSON calls the callback function it will send the data received to it.

Showing results

* Html tags in javascript
  + Use \ at the end of the line
  + `` \\ backtick, Template literal, if statement is now allowed in a template literal but ternary operator (short if with ? ) is.
* How to loop through in string literal
  + Use **map function** // map is used to create a new version of the array, run a function once for each array, the map uses a new version of the array that uses list item ['<li></li>',…], it output itself with a **comma,** so use join
* **Relative addressing**. some sites are not installed at the root of the domain using '/wp-json/wp/v2/posts?search=' will cause a problem so use
* **wp\_localize\_script**(name of js file,2,3) // in wp\_enqueue\_scripts. It lets to output some javascript data into the html source of the web page. The name of the js file is the handle that you defined as the first argument of wp\_enqueue\_script

getResults(){

$.getJSON(universityData['root\_url']+'/wp-json/wp/v2/posts?search='+this.searchField.val(),(posts)=>{

// if function is used 'this' is going to refer to getJSON because it calls the anonymous function but arrow is okay

// if you do posts.length != 0, it doesn't work.

this.resultsDiv.html(`

<h2 class="search-overlay\_\_section-title">General Information:</h2>

${posts.length ? `<ul class="min-list link-list">` : '<p>No general information that matches that search</p>'}

${posts.map( post => `<li><a href=${post['link']}>${post['title']['rendered']}</a></li>`).join('')}

</ul>

${posts.length ? '</ul>' : '' }

`);

this.isSpinnerVisible = false;

})}

Html of the search in JS

* Create a method

addSearchHTML(){

$('footer').after(`

<div class="search-overlay">

<div class="search-overlay\_\_top">

<div class="container">

<i class="fa fa-search search-overlay\_\_icon" aria-hidden="true"></i> <!-- aria-hidden is for when someone doesn't have good vision comes to our website, screen reader won't read the element -->

<input type="text" class="search-term" placeholder="What are you looking for?" id="search-term" >

<i class="fa fa-window-close search-overlay\_\_close" aria-hidden="true"></i>

</div>

</div>

<div class="container">

<div id="search-overlay\_\_result"></div>

</div>

</div>

`)

//this also works. $('body').append(``)

Retrieve info from other urls

* Since the callback is not called until the server returns the data we can do something like nested getJSON methods in callbacks because we can aggregate the data and wait until each one is received.

$.getJSON(universityData['root\_url']+'/wp-json/wp/v2/posts?search='+this.searchField.val(),(posts)=>{

$.getJSON(universityData['root\_url']+'/wp-json/wp/v2/pages?search='+this.searchField.val(), pages =>{

let combinedREsults = posts.concat(pages)

this.resultsDiv.html(`

<h2 class="search-overlay\_\_section-title">General Information:</h2>

${combinedREsults.length ? `<ul class="min-list link-list">` : '<p>No general information that matches that search</p>'}

${combinedREsults.map( post => `<li><a href=${post['link']}>${post['title']['rendered']}</a></li>`).join('')}

</ul>

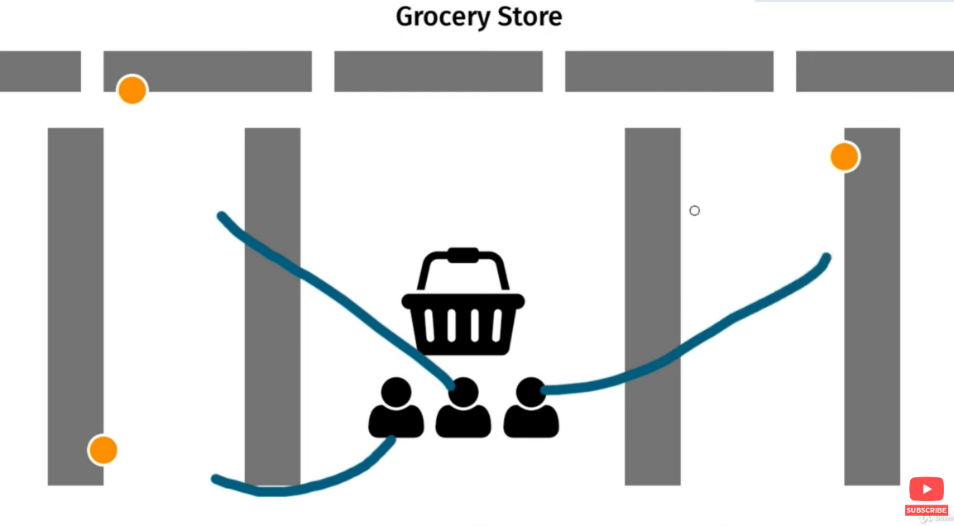
${combinedREsults.length ? '</ul>' : '' }

`);

this.isSpinnerVisible = false;

}) })

* But the nested getJSONs are not good practice: UX
  + Synchronous
  + Asynchronous



* Do both request async, If both requests are finished you can show the result.
  + When().then() // method of jquery, all requests should be completed! So then is executed

**$.when**(

//since when is babysitting the getJSON we don't need the callback function but the data returned is not actual wanted json data, but with additional information about the request, the first element is the data

$.getJSON(universityData['root\_url']+'/wp-json/wp/v2/posts?search='+this.searchField.val()),

$.getJSON(universityData['root\_url']+'/wp-json/wp/v2/pages?search='+this.searchField.val())

).**then((posts, pages)** => {

**let combinedREsults** = **posts[0]**.concat(**pages[0]**)

this.resultsDiv.html(`

<h2 class="search-overlay\_\_section-title">General Information:</h2>

${combinedResults.length ? `<ul class="min-list link-list">` : '<p>No general information that matches that search</p>'}

${combinedResults.map( post => `<li><a href=${post['link']}>${post['title']['rendered']}</a></li>`).join('')}

</ul>

${combinedResults.length ? '</ul>' : '' }

`);

this.isSpinnerVisible = false;

}, ()=>{

this.resultsDiv.html('<p>Unexpected error; Please try again</p>')

})

* + Using await

Custom Rest API

* Rest api for custom post type: <http://localhost/wreact/wp-json/wp/v2/professor>
  + Add show\_in\_rest // the **built-in** rest api endpoints, that we can readily use

if(!this.isSpinnerVisible){

this.resultsDiv.html('<div class="spinner-loader"></div> ')

this.isSpinnerVisible = true

}…

Why?

* Customize search logic // Because wordpress search, for example, is not advanced, it looks into obvious fields, the title, and the body not the custom fields and relationships. So we can do this by custom WordPress url.
* Respond with fewer json data // in search for example we don't need all.
* Send fewer getJSON requests
* Perfect exercise to sharpen php skills

How

* We can create a new folder(inc) to stay organized and then include it in function.php
  + **Require get\_theme\_file\_path('/inc/search-route')**
  + **add\_action('rest\_api\_init', 'universityRegisterSearch');** // in the new file
* **register\_rest\_route('name','rout',array())**
  + wp namespace in the rest is for core WordPress
  + Route // is the ending part of the URL
  + /v1/ // Without changing much to change the versions
  + Array // what happens when someone visits this url.
  + method // The CRUD, what they wanted to do with the url.
  + GET // always works but use this to make sure of all hosts out there **WP\_REST\_SERVER::READABLE**, it is a constant that will substitute it with get but based on the server may use a slightly different value than get. It's safer.
    - Callback // what it returns it's the JSON data that we need.

function universityRegisterSearch(){

register\_rest\_route('university/v1','search',array(

'method'=>'WP\_REST\_SERVER::READABLE',

'callback' => 'universitySearchResults'

));

}

function universitySearchResults(){

return 'congrats!';

}

* We don't need to stress about JSON syntax, we can return php and wordpress does the hard work for us. It will convert php data structures into valid data json for us. For example
  + array('red', 'orange', 'yellow') // this valid php code but not valid javascript, or JSON, but WordPress automatically converts it when is sent as a response. This is good because we can use the custom query and other functions.

function universitySearchResults(){

$professors = new WP\_Query(array(

'post\_type' => 'professor'

));

//$professors->posts, this is what we can return

// to only return a few we use array with while

$professorResults = array();

while($professors->have\_posts()){

// this we had access to the posts

$professors->the\_post();

array\_push($professorResults,array(

'id' => get\_the\_ID(),

'title' => get\_the\_title(),

'content' => get\_the\_content(),

'permalink' => get\_the\_permalink()));

}

return $professorResults;

}

Route for search

* wordpress wp\_query has another argument named, s, search.
* Wordpress passes the data in url to the callback function.
* **sanitize\_text\_field()** => WordPress has its security check, that's why you can't simply search for content on other websites, but we can be extra careful.

$professors = new WP\_Query(array(

'post\_type' => 'professor',

's' => sanitize\_text\_field($data['term'])));

All the post types

* We can use an array for post\_type // This will mix them

$professors = new WP\_Query(array(

'post\_type' => array('post', 'page', 'professor'),

's' => sanitize\_text\_field($data['term'])));

* To separate them

$results = array(

'generalInfo' => array(),

'professors' => array(),

'programs' => array(),

'events' => array(),

'campuses' => array() );

while($mainQuery->have\_posts()){

// this we had access to the posts

$mainQuery->the\_post();

if(get\_post\_type() == 'post' OR get\_post\_type() == 'page'){

array\_push($results['generalInfo'],array(

'id' => get\_the\_ID(),

'title' => get\_the\_title(),

'permalink' => get\_the\_permalink(),

'postType' => get\_post\_type(),

'authorName' => get\_the\_author()

)); }

if(get\_post\_type() == 'professor'){

array\_push($results['professors'],array(

'id' => get\_the\_ID(),

'title' => get\_the\_title(),

'permalink' => get\_the\_permalink()));}

…­­

To show custom search in js

$.getJSON(universityData['root\_url']+'/wp-**json/university/v1/search**?**term**='+this.searchField.val(), (**results**) => {

this.resultsDiv.html(`

<div class="row">

<div class="one-third">

<h2 class="search-overlay\_\_section-title">General Information</h1>

${**results['generalInfo'].**length ? `<ul class="min-list link-list">` : '<p>No general information that matches that search.</p>'}

${results['generalInfo'].map( item => `<li><a href=${item['permalink']}>${item['title']}</a> ${item['postType'] == 'post'? `by ${item['authorName']}`: ''} </li>`).join('')}

${results['generalInfo'].length ? '</ul>' : '' }

</div>

<div class="one-third">

<h2 class="search-overlay\_\_section-title">Programs</h1>

${**results['programs']**.length ? `<ul class="min-list link-list">` : `<p>No programs match that search. <a href = "${universityData['root\_rul']}/programs">View all programs</a></p>`}

${results['programs'].map( item => `<li><a href=${item['permalink']}>${item['title']}</a></li>`).join('')}

${results['programs'].length ? '</ul>' : '' }

<h2 class="search-overlay\_\_section-title">Professors</h1>

</div>

<div class="one-third">

<h2 class="search-overlay\_\_section-title">Campuses</h1>

${**results['campuses']**.length ? `<ul class="min-list link-list">` : `<p>No campuses match that search. <a href = "${universityData['root\_rul']}/campuses">View all campuses</a></p>`}

${results['campuses'].map( item => `<li><a href=${item['permalink']}>${item['title']}</a></li>`).join('')}

${results['campuses'].length ? '</ul>' : '' }

<h2 class="search-overlay\_\_section-title">Events</h1>

</div>

</div>

`);

this.isSpinnerVisible = false;

});

Images in custom API

// zero means the current post

'postThumbnail' => get\_the\_post\_thumbnail\_url(0, 'professorLandscape')

Info from custom field

if (get\_post\_type() == 'event') {

$eventDate = new DateTime(get\_field('event\_date', false, false));

$description = null;

if (has\_excerpt()) {

$description = get\_the\_excerpt();

} else {

$description = wp\_trim\_words(get\_the\_content(), 18);

}

array\_push($results['events'], array(

'id' => get\_the\_ID(),

'title' => get\_the\_title(),

'permalink' => get\_the\_permalink(),

'month' => $eventDate->format('M'),

'day' => $eventDate->format('d'),

'description' => $description

));

Search within custom fields (relationship)

* Think backward. Reverse engineer.
* Use a custom query
* **array\_unique(a,b)** // To remove duplicates if the body has the term use, it returns each item as a value of index key (0=>{}, 1=>{}, …).
  + a // the array
  + B // SORT\_REGULAR, to tell play nicely with associative arrays. Look within each subarray of the array and look for duplicates.
  + **array\_values()** // to remove the added keys
* How to check for multiple ids in serialized value?
  + Use multiple arrays for 'meta\_query'
  + Set the relationship of those multiple arrays to OR, default is AND
* Invalid search term that returns all of the prof
  + If the query is empty, it returns all the profs
* What if the program body has a word from another program? This is not what we want. So in programs, we don't want to look to the body, which can throw false positives.
  + **Hand-crafted SQL**
  + Put the body in **another field** // hide the other **wysiwyg** editor by removing the support editor in the **must-use plugin**.

if ($results['programs']) {

**$meta\_query** = array('relation' => 'OR');

foreach **($results['programs']** as $item) { // loop through them directly

array\_push($meta\_query, array(

'key' => 'related\_programs',

'value' => '"'.$item['id'].'"',

'compare' => 'LIKE'

));

}

$programRelatedProf = new WP\_Query(array(

'posts\_per\_page' => -1,

'post\_type' => 'professor',

'orderby' => 'title',

'order' => 'ASC',

'meta\_query' => **$meta\_query**

));

while ($programRelatedProf->have\_posts()) {

$programRelatedProf->the\_post();

array\_push($results['professors'], array(

'id' => get\_the\_ID(),

'title' => get\_the\_title(),

'permalink' => get\_the\_permalink(),

'postThumbnail' => get\_the\_post\_thumbnail\_url(0, 'professorLandscape')));

}

$results['professors'] = array\_values(array\_unique($results['professors'], SORT\_REGULAR));

}

How to make things work for other relationships

* Use the same query with an array of post\_types.
  + Events can be added to the professor query
* the relationship for campuses is inside the program, because of the fewer number of campuses compared to the program
  + We have access to the campus of a program when we are pulling program data, since **get\_field()** gives the true information of the related campus like id, it is an object, and we can give the object as an argument to **get\_the\_title().** Custom meta\_query doesn't have the power of get\_field().

Non-js fallback traditional WordPress searching

* <http://localhost:3000/wreact/?s=biology>
  + Powered by index.php
  + We need search.php in the theme folder
* Create a page in admin named search
* Create a template file named **page-search.php** to support it. This will hold the form.
* **esc\_url(site\_url())** // when manually echoing a url that comes from a database, for example, write it before site\_url(). To protect our users when we are hacked.
* We want to use the default search url of WordPress so we use the above address for the action of the form tag.
* a tag // for the search icon to go to the search address.
  + **return false** // for openOverlay, prevents the default behavior of the link element, and if javascript is enabled, it will run. When clicking on the icon, this function is called.

<div class="generic-content">

<form class="search-form" method="get" action="<?php echo esc\_url(site\_url('/')) ?>">

<!-- to help with accessibility we add the label, if it is clicked the input will be focused, for is for id -->

<label class="headline headline--medium" for="s">Perform a new search:</label>

<div class="search-form-row">

<input class="s" type="search" name="s" id="s" placeholder="What are you looking for?">

<input class="search-submit" type="submit" value="Search">

</div>

</form>

</div>

* Search.php // results of normal search
* **get\_search\_query()** // get the value of the searched item (as in url)
  + **cross-site scripting attack** // get\_search\_query() escapes value by default and takes even scripts as text, don't run them
  + **get\_search\_query('false')** // overrides the default and executes javascript
  + <http://localhost:3000/wreact/?s=%3Cscript%3Ealert%28%22haha%22%29%3C%2Fscript%3E> // it will run the script! Google Chrome may prevent it
  + It's good when we want to use it as the value of an input, but outputting as the html may not be a good option.
* **esc\_html()**

esc\_html(get\_search\_query(false))

different template for each post type in the normal search page

* if statement
* get\_template\_part()
* if the search query have\_post() then loop
* **searchform.php** // search form in search.php, WordPress looks for a file named searchform.php in the root, instead of duplicate we can use this.
  + **get\_search\_form()**

<?php

if(have\_posts()){

while (have\_posts()) {

the\_post();

// instead of hardcoding based on post type get the related tmeplate

get\_template\_part('template-parts/content', get\_post\_type());

}

echo paginate\_links();

}else{

echo '<h2n class="headline headline--small-plus">No Results match that search.</h2>';

}

get\_search\_form()

?>

Default wordpress user roles

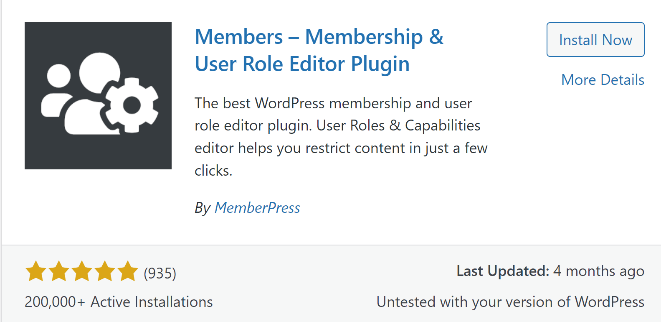
* users
* subscriber // only manage their own profile info
* contributor // write post drafts
* Author // Can write and publish content, but they can not change someone elses content
* Editor // Create and edit other content, all post types
* Administrator // Whatever they want to do.

To login as two different users

* Use incognito // new private window

How to give access to only one post type

* Custom roles
  + Members // plugin



Members' specific post type

* By default, the plugin offers all post types as one to add to the role
* **'capability\_type' => 'event'** // in register\_post\_type. default is the post which means if you have permission to edit posts you have permission to edit events.
* **'map\_meta\_cap' => true** // To enforce the permission set above, if not set we have to set the logic when want to use the capability. For example, without this, we can set much logic for permissions. This will give us a reasonable setting.
* **T**hese two options will require us to permit our users. The event will not appear on our user admin anymore, we have to set it.

Multiple Roles to a single user

* The members allow it. Combine roles and assign based on the user

 Open Registration (CRUD user and security)

* In setting-> general-> anyone subscribe
* [http://localhost:3000/wreact/wp-signup.php //](http://localhost:3000/wreact/wp-signup.php%20//) and wp-login.php

href="<?php echo **esc\_url(site\_url('/wp-signup.php'))** ?>"

Login and registeration functions

* **wp\_login\_url()** // we can instead use this
* **wp\_logout\_url()**

<a href="<?php echo wp\_login\_url() ?>" class="btn btn--small btn--dark-orange float-left">Log Out</a>

* **wp\_registration\_url()**

Redirect users

* Wordpress will email the password, the local dev environment may not set the new password in the admin.
* It **redirects** to the admin dashboard but they can't do anything there so redirect to the website
* To **redirect**
  + **admin-init** // add action hook

//redirect subscriber accounts out of admin and onto homepage

add\_action(**'admin\_init'**, 'redirectSubsToFrontend');

function redirectSubsToFrontend(){

$ourCurrentUser = **wp\_get\_current\_user()**;

//roles is an array

if(count($ourCurrentUser**->roles**) == 1 AND $ourCurrentUser->roles[0] == 'subscriber'){

**wp\_redirect(site\_url('/'))**;

// tell wordpress stop spinning its gears when redirecting

**exit**; }}

* + **wp\_get\_current\_user()->roles**
  + **wp\_redirect(site\_url('/'))**
* **get\_avatar(user id, avatar size)** // shows the user avatar, must use **echo**
  + **get\_current\_user\_id()**

get\_avatar(get\_current\_user\_id(), 60)

* gravatar //globally recognized avatar

Hide admin nav bar

* **show\_admin\_bar(false);**
* **wp\_loaded** // add action hook

add\_action('wp\_loaded ', 'noSubsAdminBar');

function noSubsAdminBar(){

$ourCurrentUser = wp\_get\_current\_user();

//roles is an array

if(count($ourCurrentUser->roles) == 1 AND $ourCurrentUser->roles[0] == 'subscriber'){

**show\_admin\_bar(false);**

}

}

Login page appearance

* login logo url

//customize login screen

add\_filter('login\_headerurl', 'ourHeaderUrl');

function ourHeaderUrl(){

return esc\_url(site\_url('/'));}

* By default, wordpress **does not load theme css for login. Because they are considered front-end**
* We have to override css
  + **add\_action('login\_enqueue\_scripts','')** // hooks for login
  + **add\_filter('login\_headertitle','')** // filter hook for login logo name
  + **get\_bloginfo('name')**

add\_action(**'login\_enqueue\_scripts'**, 'ourLoginCSS');

function ourLoginCss(){

**wp\_enqueue\_style**('site', get\_stylesheet\_uri());

wp\_enqueue\_style('custom-font', '//fonts.googleapis.com/css?family=Roboto+Condensed:300,300i,400,400i,700,700i|Roboto:100,300,400,400i,700,700i'); //name does not matter

}

add\_filter(**'login\_headertitle'**,'ourLoginTitle');

function ourLoginTitle(){

//return what you want to show

return **get\_bloginfo('name')**;

}

* custom login template // See <https://youtu.be/HafkLf1EPdw>

Note API (CRUD)

* create a page in admin // To have an address
* page-my-notes.php
* Only loggedin => check in header and page-my-notes.php(redirect)
* Save on WordPress database
  + Note post type

register\_post\_type('note', array(

'show\_in\_rest'=> true,

'supports' => array('title', 'editor'),

// don't show it in public queries or search result only, specific to user account, it also hides in dashboard

'public' => false,

// to show it dashboard

'show\_ui' => true,

'labels' => array(

'name' => 'note',

'add\_new\_item' => 'Add New Note',

'edit\_item' => 'Edit Note',

'all\_items' => 'All Notes',

'singular\_name' => 'Note'

),

'menu\_icon' => 'dashicons-welcome-write-blog' ));

<?php $userNotes = new WP\_Query(array(

'post\_type' => 'note',

'posts\_per\_page' => -1,

// only current user notes

'author' => get\_current\_user\_id()));

while($userNotes->have\_posts()){

$userNotes->the\_post(); ?>

<li>

<input class="note-title-field" value="<?php echo esc\_attr(get\_the\_title()); ?>">

<span class="edit-note"><i class="fa fa-pencil" aria-hidden="true"></i> Edit</span>

<span class="delete-note"><i class="fa fa-trash-o" aria-hidden="true"></i> Delete</span>

<textarea class="note-body-field" ><?php echo esc\_attr(strip\_tags(get\_the\_content())) ?></textarea>

</li>

<?php

} ?>

The query

Javascript

import $ from 'jquery' // jquery must be in ''

class MyNotes {

constructor(){

this.event()

}

event(){

$('.delete-note').on("click", this.deleteNote.bind(this))

}

//methods will go here

deleteNote(){

alert('hi')

}

}

export default MyNotes;

HTTP Requests

* Get // <http://localhost:3000/wreact/wp-json/wp/v2/note/166>
* Post
* Delete

$.ajax({

url: universityData.root\_url + '/wp-json/wp/v2/note/166',

type: 'DELETE',

// if the request is sucessful

success: (response)=>{

console.log("congrats")

console.log(response)

},

error: (response)=>{

console.log("Sorry")

console.log(response)

}

});

}

Nonce (number used once)

* A little piece of secret data
* Whenever logged in account, wordpress generates a nonce
* wp\_localize\_script() // to send the nonce to be available in js

wp\_localize\_script('main-university-js', 'universityData', array(

'root\_url'=> get\_site\_url(),

'nonce' => wp\_create\_nonce('wp\_rest')

));

}

* beforeSend //Change the header
* 'X-WP-Nonce'

beforeSend: (xhr)=>{

//modify the request

xhr.setRequestHeader('X-WP-Nonce', universityData.nonce);

},

* in Html li element:

data-id="<?php the\_ID()?>"

* How to get the id
  + thisNote.data('id')

editNote(e){

let thisNote = $(e.target).parents('li')

…

url: universityData.root\_url + '/wp-json/wp/v2/note/' + **thisNote.data('id')**, //doesn't work with .dataset['id']

* appearance
  + readonly // in html input tag
  + edit to make inputs editable
  + save button which is hidden when the edit is hit becomes available

<span class="update-note btn btn--blue btn--small"><i class="fa fa-arrow-right" aria-hidden="true"></i> Save</span>

* js

editNote(e){

let thisNote = $(e.target).parents('li')

if(thisNote.data("state") == 'editable'){

//don't forget bind

this.makeNoteReadOnly(thisNote)

} else{

this.makeNoteEditable(thisNote)

}

}

makeNoteEditable(thisNote){

thisNote.find(".edit-note").html('<i class="fa fa-times" aria-hidden="true"></i> Cancel')

thisNote.find('.note-title-field, .note-body-field')

.removeAttr("readonly")

.addClass("note-active-field");

thisNote.data("state", "editable")

thisNote.find('.update-note').addClass("update-note--visible");

}

makeNoteReadOnly(thisNote){

thisNote.find(".edit-note").html('<i class="fa fa-pencil" aria-hidden="true"></i> Edit')

thisNote.find('.note-title-field, .note-body-field')

.attr("readonly", "readonly") //name and value

.removeClass("note-active-field");

thisNote.data("state", "cancel");

thisNote.find('.update-note').removeClass("update-note--visible");

}

* Update

updateNote(e){

let thisNote = $(e.target).parents('li')

var ourUpdatedPost ={

'title': thisNote.find('.note-title-field').val(),

'content': thisNote.find('.note-body-field').val()

}

// for other https requests

$.ajax({

beforeSend: (xhr)=>{

//modify the request

xhr.setRequestHeader('X-WP-Nonce', universityData.nonce);

},

url: universityData.root\_url + '/wp-json/wp/v2/note/' + thisNote.data('id'), //doesn't work with .dataset['id']

type: 'POST',

data: ourUpdatedPost,

// if the request is sucessful

success: (response)=>{

this.makeNoteReadOnly(thisNote);

},

error: (response)=>{

console.log("Sorry")

console.log(response)

}

});

}

Create note

* Address the inputs directly
* **.prependTo()** // to add the html
* **$('#my-notes').on("click", ".delete-note", this.deleteNote)** // The buttons created after we set the events so by using this method, where ever we click on the parent and the interior element has delete-note, then the function is triggered.

Permissions on note post type

* Try to use the other incognito and without workflow url, since ajax may send a request to all open urls.

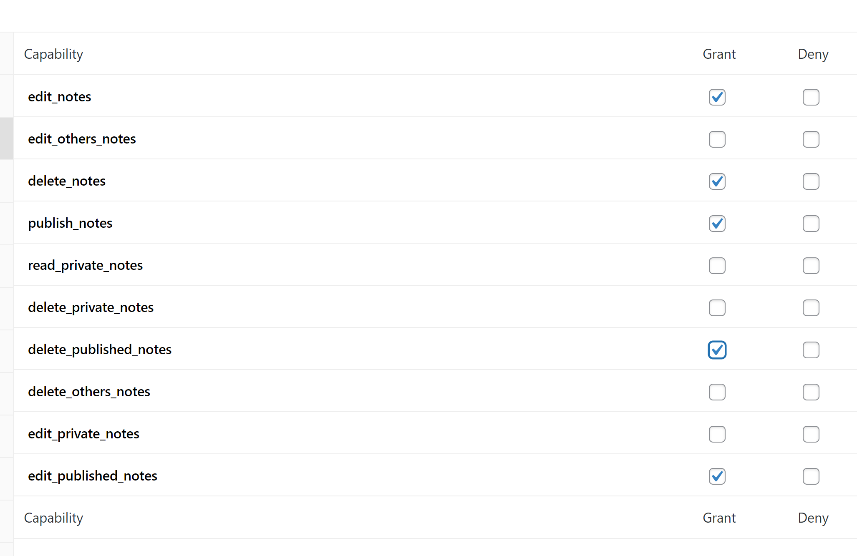
// we don't want to give them access to all posts just note

register\_post\_type('note', array(

'capability\_type' => 'note',

'map\_meta\_cap' => true,

* For subscriber:



Privacy (private content)

* Rest api // it shows all notes to anyone, Only the creator of the note should be able to view the note
* Status: 'private' // javascript is easily hacked
* In functions.php // filter the data before saving them into the database

//force note posts to be private

// the hook is one of them most powerful functions.

add\_filter(**'wp\_insert\_post\_data'**, 'makeNotePrivate');

function makeNotePrivate($data){

if($data['post\_type'] == **'note'** and $data['post\_status'] != **'trash'**){

$data['post\_status'] = 'private';

}

return $data;

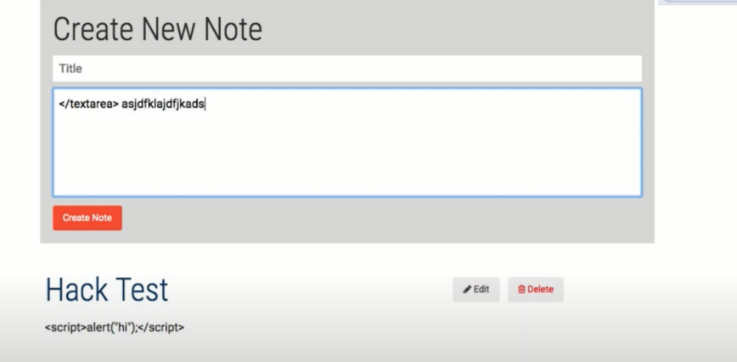
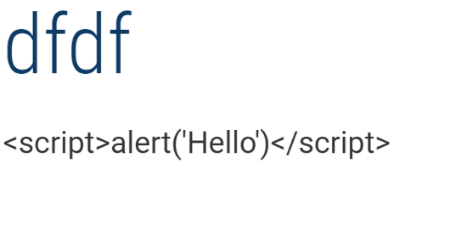
}

* Private word is shown in the front end
* str\_replace('Private: ', '', esc\_attr(get\_the\_title()));



Security of input data

* Only the admin account needs to be able to enter unfiltered html



* Use **esc\_functions**
  + esc\_html()
  + esc\_textarea()
  + esc\_atrr()
* Strict in server-side // no html()
* 'wp\_insert\_post\_data' hook
* sanitize\_()
  + sanitize\_textarea\_field() // removes the html tags
  + sanitize\_text\_field // removes the html tags

if($data['post\_type'] == 'note'){

$data['post\_content'] = sanitize\_textarea\_field($data['post\_content']);

$data['post\_title'] = sanitize\_text\_field($data['post\_title']);

}

Post limit

* wp\_insert\_post\_data
* count\_user\_posts(user id, post type)

if($data['post\_type'] == 'note'){

if(count\_user\_posts(get\_current\_user\_id(),'note') > 4 ){

die("You have reached your note limit!");

}…

* it effect the delete option, since the if does not allow anything, we can differentiate create request by post id, since **id of new post does not exist**
* **$post\_arr** // wp\_insert\_post\_data does not pass the **id** by $data - first argument, we need a second argument
  + 10 is for priority of call back function, multiple functions to the same hook
  + 2 is for

add\_filter('wp\_insert\_post\_data', 'makeNotePrivate', 10, 2 );

if($data['post\_type'] == 'note'){

if(count\_user\_posts(get\_current\_user\_id(),'note') > 4 AND $postarr['ID'] ){

die("You have reached your note limit!");

}…

Hide not limit message based on the count

* 'rest\_api\_init' // action hook, add a new property to the rest API response

function university\_custom\_rest(){

register\_rest\_field('post','authorName',array(

'get\_callback' => function() {return get\_the\_author();}

));

register\_rest\_field('note','userNoteCount',array(

'get\_callback' => function() {return count\_user\_posts(get\_current\_user\_id(), 'note');}

));

}

add\_action('rest\_api\_init', 'university\_custom\_rest');

(Custom endpoints) Likes

* It has a different logic,
  + For post prof
  + Only once
  + Tied to user id
* Create a post,
* Liked Professor ID // Number time, Create a custom field that has the professor's id
* Custom endpoints
* Query-> found\_posts // a property that gives the absolute number of posts found, all posts found with the given condition.
* Fill in the heart-based user who liked //
  + data- = 'yes'
  + 'author' => get\_current\_user\_id() // another query, it is not logged in return 0 and it's like we don't filter author, so we check if the user is logged in.
  + $existsQuery // A var

register\_post\_type('like', array(

'supports' => array('title'),

'public' => false,

'show\_ui' => true,

'labels' => array(

'name' => 'Likes',

'add\_new\_item' => 'Add New Like',

'edit\_item' => 'Edit Like',

'all\_items' => 'All Likes',

'singular\_name' => 'Like'

),

'menu\_icon' => 'dashicons-heart'

));

* The html

$likeCount = new WP\_Query(array(

'post\_type' => 'like',

// pull only for this prof

'meta\_query' => array(

array(

'key'=> 'liked\_professor\_id' ,

'compare' => '=',

'value' => get\_the\_ID()

))));

$existStatus = 'no';

if(is\_user\_logged\_in()){

$existQuery = new WP\_Query(array(

'author' => get\_current\_user\_id(),

'post\_type' => 'like',

'meta\_query' => array(

array(

'key'=> 'liked\_professor\_id' ,

'compare' => '=',

'value' => get\_the\_ID()

))));

$existQuery->have\_posts() ? $existStatus = 'yes' : '';

} ?>

<span class="like-box" data-like="<?php echo $existQuery->posts[0]->ID; ?>" data-professor="<?php the\_ID() ?>" data-exists="<?php echo $existStatus ?>">

<i class="fa fa-heart-o" aria-hidden="true"></i>

<i class="fa fa-heart" aria-hidden="true"></i>

<span class="like-count"><?php echo $likeCount->found\_posts; ?></span>

</span>

Js

* data-exists // to see whether current users have liked or not

import $ from 'jquery'

//Don't forget the jquery!

class Like{

constructor(){

this.events();}

events(){

$(".like-box").on('click', this.ourClickDispatcher.bind(this) );

}

//methods

ourClickDispatcher(e){

//make it flexible look for the event element, closest ancestor

let currentLikeBox = $(e.target).closest('.like-box')

//use attr instead of data

if(currentLikeBox.attr('data-exists') == 'yes'){

//pass the value to get the correct data value

this.deleteLike(currentLikeBox);

}else{

this.createLike(currentLikeBox); }}

createLike(currentLikeBox){

$.ajax({

beforeSend: (xhr) => {

xhr.setRequestHeader('X-WP-Nonce', universityData.nonce)

},

url: universityData.root\_url + '/wp-json/university/v1/manageLike',

type:'POST',

data: {

'professorId': currentLikeBox.data('professor')

},

success: (response) => {

// data does not update you should use attr

currentLikeBox.attr('data-exists','yes');

let likeCount = parseInt(currentLikeBox.find('.like-count').html(), 10) //base 10

likeCount++

currentLikeBox.find('.like-count').html(likeCount);

// only response since it returns only the id

currentLikeBox.attr('data-like', response);

console.log(response);

},

error: (response) => {

if(response['responseText'] == 'Invalid Professor id'){

// $('.like-count').html(int($('.like-count').html) + 1)

}

console.log(response)

}});}

* delete function

deleteLike(currentLikeBox){

$.ajax({

beforeSend: (xhr) => {

xhr.setRequestHeader('X-WP-Nonce', universityData.nonce)

},

url: universityData.root\_url + '/wp-json/university/v1/manageLike',

type:'DELETE',

data: {

'like': currentLikeBox.attr('data-like')

},

success: (response) => {

currentLikeBox.attr('data-exists','no');

let likeCount = parseInt(currentLikeBox.find('.like-count').html(), 10)

likeCount--

currentLikeBox.find('.like-count').html(likeCount);

currentLikeBox.attr('data-like', '');

console.log(response);

},

error: (response) => {

alert('ha')

console.log(response)

}});}

Custom endpoints

* **the built-in apis** //
* **'rest\_api\_init'** // Hooke
* **register\_rest\_route(a,b,c)**

Create

* **wp\_insert\_post(array( ));** // return the id of the post that it created
* **'meta\_input' => array(metafiled key name => meta filed value)** //WordPress native custom fields, WordPress advanced field plugin is just an abstraction layer between WordPress native fields and gives us a luxurious intuitive interface so we still can use this.
* WordPress interprets the data property of the Ajax function as if the item is written in url
* Permission
  + **current\_user\_can('publish\_note')**
  + **is\_user\_logged\_in()** // evaluates false if we don't use the nonce

if(is\_user\_logged\_in()){

$professor = sanitize\_text\_field($data['professorId']);

$existQuery = new WP\_Query(array(

'author' => get\_current\_user\_id(),

'post\_type' => 'like',

'meta\_query' => array(

array(

'key'=> 'liked\_professor\_id',

'compare' => '=',

'value' => $professor ))));

if($existQuery->found\_posts == 0 and get\_post\_type($professor) == 'professor'){

return wp\_insert\_post(array(

'post\_type' => 'like',

'post\_status' => 'publish',

'post\_title' => 'Our php',

'post\_content' => 'body',

'meta\_input' => array( //**wordpress native custom fields**

'liked\_professor\_id' => $professor )));

}else{

die('Invalid Professor Id');

}

} else{

die('Only logged in users can create a like');

}

* $existQuery->**found\_posts** == 0 //user already liked?, **found\_post** is a **property**
* **get\_post\_type($professor) == 'professor'** // is it really professor type

delete

* **data-like**="<?php echo $existQuery->posts[0]->ID ?>" // for delete
* **data()** // only runs and gets the value only once when the first-page load so in ourClickDispatcher when we want to toggle use **attr()** instead.
* **wp\_delete\_post(id, true if skip trash);**
  + Check user permission if the like belongs to the user
  + If the post is a type like
* **return 'Congrats like deleted'** // if you don't right the return the success of js will not run! the error will run!

function deleteLike($data) {

$likeId = sanitize\_text\_field($data['like']);

//set permission check, if the like for the current user and the post is like

if(get\_current\_user\_id() == get\_post\_field('post\_author', $likeId) AND get\_post\_type($likeId) == 'like' ){

wp\_delete\_post($likeId, true);

// if you don't right the return the success of js will not run! the error will run!

return 'Congrats like deleted';

}else{

die("You do not have permission to delete that.");

}

}

* **Isset()**

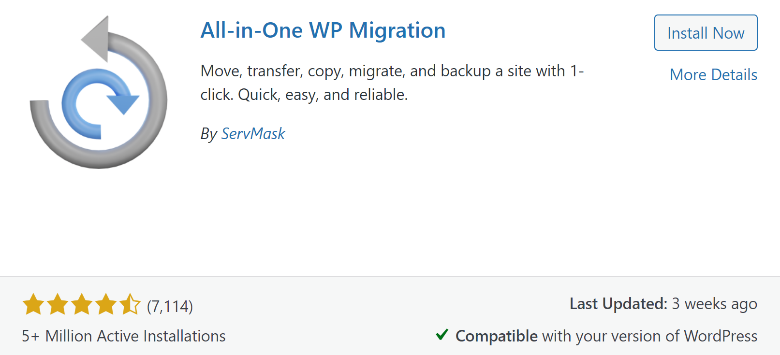
Hosting the site (Deploying)

Web host

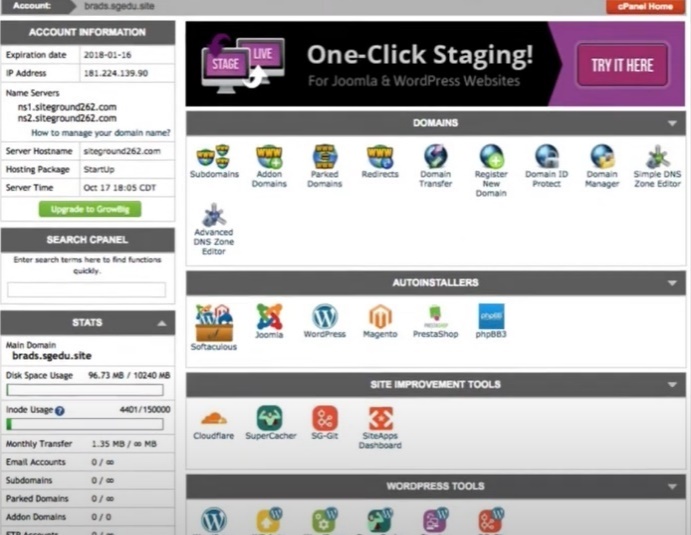
* site ground // suggestion of WordPress itself
* 000webhost.com // use this

Option 1 (Plugin)

* The import part of the WordPress // exports only files
* All-in-one wp migration // plugin

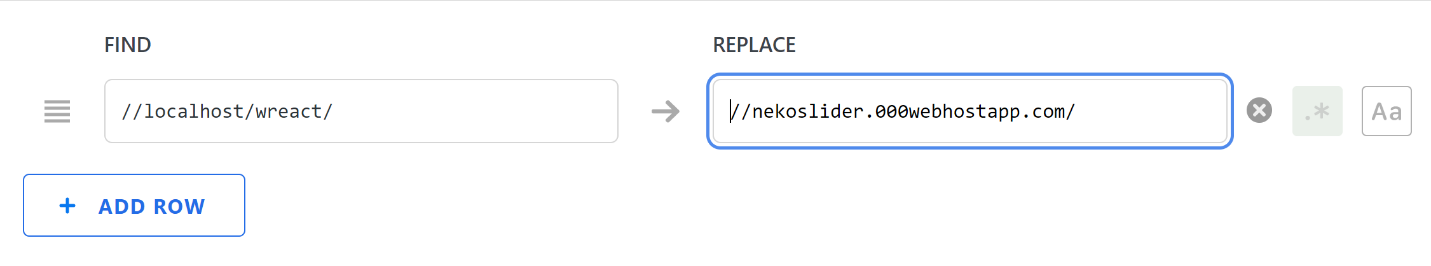


* Export -> files // database, theme files, plugins, photographs uploaded
* Install WordPress on host -> install the plugin -> import -> permalink cache (save twice)
* Cpanel // changes in files -> file manager



Option 2 (Git)

* Future updates are easy (only push to repo)
* No need for FTP
* Database => backup
  + Heidi
  + Copy, find, and replace // in WordPress this is needed for database
    - Domain url // all records should be replaced with new
      * Manually
      * **wp migrate db** // WordPress admin, export, and replace



* create a database for hosting
* then use php my admin in hosting to upload it.
  + Database credentials
    - **In wp-config.php**
    - **a way that works both for local and host**
      * create local.php in the root
      * set an if statement for the file
      * exclude the file in git to be uploaded in git ignore, if you already added it you have to remove it from there create the git.ignore, and then add and commit again

if (file\_exists(dirname(\_\_FILE\_\_).'/local.php')) {

//loca databse settings.

define('DB\_NAME', 'wreact');

define('DB\_USER', 'wreact');

define('DB\_PASSWORD', '1111');

define('DB\_HOST', 'localhost');

} else {

//Live database settings

define('DB\_NAME', 'id20085101\_universitydata');

define('DB\_USER', ' id20085101\_wreact');

define('DB\_PASSWORD', '5ID\*Mr/rl15c<0\*n');

define('DB\_HOST', 'localhost');

}

**Authontication**

in wp-config.php

* + set the random keys
  + using URL <https://api.wordpress.org/secret-key/1.1/salt/> // generates a random salt each time
  + bitbucket // gives free private repo

Automatic uploading of the repo to host

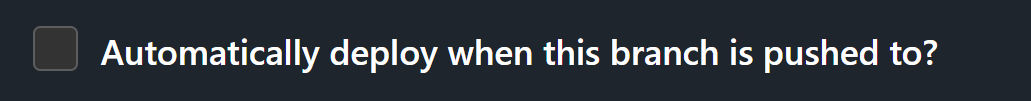
* **Third-party deployment service** // There are some services and companies that
* do this
  + **deploybot**
  + **ftploy**
  + **Deplotyhq**

Deplotyhq

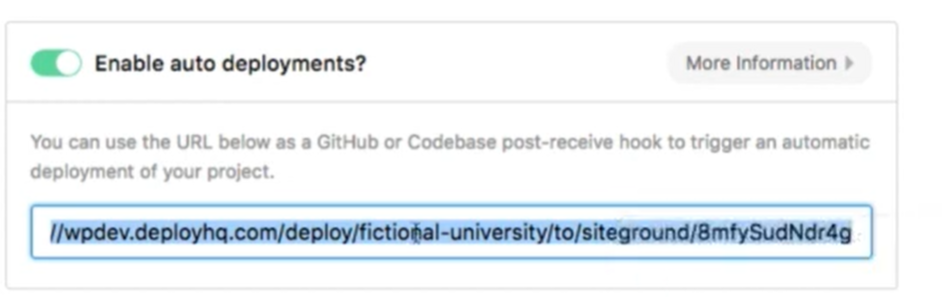
* Permit to access your repo
* Don't use ftp which sends passwords unprotected
* ssh/ssftp
* Check the hosting setting for ftp user pass info
* Check 'use ssh'
  + Copy all the generated text
  + Ssh access // In CPanel
    - Upload ssh key
* Deployment path // public\_html/
* Git **submodule error** => This occurs when the repository is using files cloned from another repository but has had no mapping reference to the source repository created for it.
  + **git rm --cached wp-content/reactpress/apps/wpreact**
  + **commit -m ""**
  + **push**
  + Clean all the files from the created repo before pushing
  + git push uni3 master –force
  + When uploading check the limitation of bandwidth, free ones don't allow long connections.
* **Connection error** // If the **file size** for **deploying** is big check your files and the backups, in D:\xampp\htdocs\wreact\wp-content\ai1wm-backups
* **Database connection error** // cpanel -> mysql databased -> add user to database (create – alter …, all privileges)

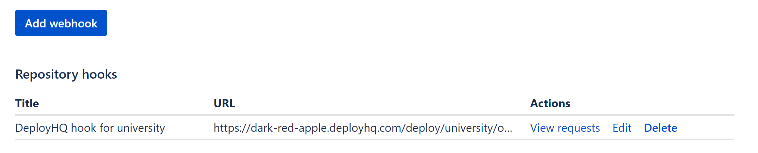
Automation

* Go to the server of the automation service



* You may need a url hook // copy url, go to GitHub or bitbucket setting, webhook. Usually automatically is added





Plugins

* Plugin folder in wp-content
* Create a folder with a php file with the same name. in the php file use the comments to create the plugin.

<?php

/\*

**Plugin Name:** my first amazing plugin

**Description:** This plugin will change your life.

\*/

?>

// adding something after content

add\_filter(**'the\_conttent'**, 'amazingContentEdits');

function amazingContentEDits($content){

$content = $content.'<p> All Content belong to content </p>';

$content = str\_replace("Lorem', '\*\*\*', $conte);

Return $content;

}

* In wordpress most things can be addressed by add\_action and add\_filter hooks.
* A plugin is like a theme function file
* A plugin is a separate sharable folder
* **Add\_menu\_page()** // to show the plugin in the menu, from WordPress documentation
* **Add\_option()**

Shortcodes

* In the admin in textarea field use **[shortcode]**
* WordPress looks for this shortcode in your theme folder and other places
* **add\_sortcut('name of shortcut', 'callbackfunciton')**

add\_shortcode('programCount', 'programCountNumber');

function programCountNumber(){

$programs = new WP\_Query(array(

'post\_type' => 'program',

));

return $programs->found\_posts;

}

* check where you want to work on
* **is\_single()** //works for post or detail screen for any post type but page and attachment
* **is\_main\_query()**
* **is\_page()**

plugin word – count

* **admin setting screen**
* **admin\_menu** // add\_action hook
* **add\_options\_page** //
* **add\_options\_page('title','the name in the menu', 'what user','url in admin should be unique','function that shows the html');**
* functions name should be unique // use **classes** instead

php classes

* **\_\_construct()** // you can put add\_action, and add\_filter in the contract
* **Callable syntax** // pass the reference of a function of the class inside the class
  + **array($this, 'adminPage') // point towards the class (or an object)**

class WordCountAndTimePlugin{

//you can put the add\_action and filter in the construct

function **\_\_construct()**

{

add\_action('admin\_menu', **array($this, 'adminPage')**);

}

function adminPage(){

// the current user can change options in wordpress

add\_options\_page('Word Count Setting','Word Count', 'manage\_options', 'word-count-setting-page', array($this,'ourHTML')); }

function ourHTML(){ ?>

<div class="wrap"><h1>Word Count Settings</h1></div> // wrap for wordpress

<?php }}

$wordCountAndTimePlugin = new WordCountAndTimePlugin();

database

* **wp\_options** // the table that holds the options
  + add the option fields to this table
* add\_action(**'admin\_init'**, array($this, 'settings')); // Register the setting before worry about the form
* **register\_setting(**'the name of setting group', 'actual name of setting', array('sanitize function', 'default value') ); gourp name will be used for nonce.
* **sanitize\_text\_field** // general WordPress function that sanitizes the input value
* **add\_settings\_field()** // begin the html input form and tie it with register\_setting
* **add\_settings\_field**('the name of the function we want to tie to', 'html label text', 'actual html', 'pageSlug', 'the section of the page to add to'); // this create the field
* **add\_settings\_section**('the name of section', 'subtitle', 'the desc html', 'page slug' );// creates a section for that page URL

function settings(){

//the name of the, this is like an api

add\_settings\_section('wcp\_first\_section', null, null, 'word-count-settings-page');

add\_settings\_field('wcp\_location', 'Display Location', array($this, 'locationHTML'),'word-count-settings-page', 'wcp\_first\_section');

register\_setting('wordcountplugin', 'wcp\_location', array('sanitize\_callback' => 'sanitize\_text\_field', 'default' => '0') ); }

function locationHTML(){ ?>

<!-- match the name we gave to add setting field -->

<select name="wcp\_location">

<option value="0">Beginning Of Post</option>

<option value="1">End of Post</option>

</select>

<?php }

* **options.php** // in the html we output for add\_options\_page
  + **settings\_fields(group field name);** // define the field group so WordPress will add the hidden html values with the nonce and action value security and **permission**. You can't save without this.
  + **do\_settings\_sections(page slug);**
  + **submit\_button()**;

function ourHTML(){ ?>

<div class="wrap">

<h1>Word Count Settings</h1>

<form **action="options.php" method="POST"**>

<?php

settings\_fields('wordcountplugin');

do\_settings\_sections('word-count-settings-page');

// wordpress save blue button

submit\_button();

?>

</form>

</div>

<?php }

* Check the options table it should be there. You have to select the option and click it.

Select the current value

* selected(true value, option value)
* checked // Wordpress function that handles checkbox
* get\_option('wcp\_location') //Wordpress doesn't access the database each time it'll call this function, calling it multiple times won't affect the performance, WordPress loads all the options from the database based on their autoload value.

<option value="0" <?php selected(get\_option('wcp\_location'), 0) ?>>Beginning Of Post</option>

<option value="1" <?php selected(get\_option('wcp\_location'), 1) ?>>End of Post</option>

* Other inputs

<input type="text" name="wcp\_headline" value = "<?php echo esc\_attr(get\_option('wcp\_headline')) ?>" placeholder="Please write your headline">

<input type="text" name="wcp\_headline" value = "<?php echo esc\_attr(get\_option('wcp\_headline')) ?>" placeholder="Please write your headline">

Pass argument to html callback func

* array at the end of add\_settings\_field
  + You can add custom or pre-defined properties

add\_settings\_field('wcp\_charactercount', 'Character Count', array($this, 'charCountHTML'), 'word-count-settings-page', 'wcp\_first\_section', **array('theName' => 'wcp\_charactercount')**);

Custom Validation logic

* Sanitize callback function

register\_setting('wordcountplugin', 'wcp\_location', array('sanitize\_callback' => array($this, 'sanitizeLocation'), 'default' => '0'));

* **add\_settings\_error('name of related option', 'slug', 'actual error');**

public function sanitizeLocation($input)

{

if ($input != '0' and $input != '1') {

//wordpress add the slug like id=slug

add\_settings\_error('wcp\_location', 'wcp\_location\_error', 'Display location must be either beginning or end');

return get\_option('wcp\_location');

}

return $input;

}

The functionality of the post

* Filter('the\_content', '') // We want to filter the post

public function ifWrap($content)

{

// 1 in get\_option is the default value when plugin is installed it is not is not in the dabase

if (is\_main\_query() and is\_single() and (get\_option('wcp\_wordcount', '1') or get\_option('wcp\_charactercount', '1') or get\_option('wcp\_readtime', '1'))) {

return $this->createHTML($content);

}

return $content;

}

public function createHtml($content)

{

$html = '<h3>' . esc\_html(get\_option('wcp\_headline', 'Post Statistics')) . '</h3><p>';

if (get\_option('wcp\_wordcount', '1') or get\_option('wcp\_readtime', '1')) {

// php function

$wordCount = str\_word\_count(strip\_tags($content));

}

if (get\_option('wcp\_wordcount', '1')) {

$html .= 'This post has ' . $wordCount . ' words. <br>';

}

if (get\_option('wcp\_charactercount', '1')) {

$html .= 'This post has ' . strlen(strip\_tags($content)) . ' characters. <br>';

}

if (get\_option('wcp\_readtime', '1')) {

$html .= 'This post will take about ' . round($wordCount/255) . ' minute(s) to read. <br>';

}

$html .= '</p>';

// both zero are string

if (get\_option('wcp\_location', '0') == '0') {

return $html . $content;

}

return $content . $html;

}

Php functions

* str\_word\_count()
* strlen()
* strip\_tags()
* round()
* (Condition) ? (Statement1) : (Statement2); // php ternary operator

Translating

* In the plugin comment section for settings:
  + **Text Domain: wcpdomain** // The name should be unique
  + **Domain path: /languages**
* **\_\_('orignal text', 'text domain')** // use this for what you want to translate

public function adminPage()

{

// manage options = the current user can change options in wordpress

// name, url

add\_options\_page('Word Count Setting', \_\_('Word Count', 'wcpdomain'), 'manage\_options', 'word-count-settings-page', array($this,'ourHTML'));

}

Loading translation files

* add\_action('init', array($this, 'languages'))
* load\_plugin\_textdomain('texdomain name', false, language folder in the plugin)

…

add\_action('init', array($this, 'languages'));

}

public function languages()

{

load\_plugin\_textdomain('wcpdomain', false, dirname(plugin\_basename(\_\_FILE\_\_)).'/languages');

}

Creating translation files.

* Create a folder named **languages** in the plugin
* Some ways of creating translation
  + **POEDIT** // a piece of software for Windows and Mac
  + **LOCO Translate** // a plugin

Loco Translate

* Whenever we use \_\_(), the plugin will look through our code
* Click on **create template** option
* **Add new language** 
  + Author option
* **Wordpres -> setting -> site language**

Security

* **esc\_html\_\_ ()** //No one can put html in translated content

Custom form and submenu for the plugin (word filter)

* If (!defined('ABSPATH')) { exit; } // Exit if accessed directly. So our plugin won't have unexpected behavior.
* add\_action('admin\_menu', array($this, 'ourMenu'));
  + **add\_menu\_page()**
* add\_menu\_page('tab title', 'menu name', 'user capability needed to view the page', 'slug', 'html function', 'icon', 'order priority');

class OurWordFilterPlugin {

function \_\_construct()

{

add\_action('admin\_menu', array($this, 'ourMenu'));

}

function ourMenu(){

add\_menu\_page('Words To Filter', 'word Filter', 'manage\_options', 'ourwordfilter', array($this, 'wordFilterPage'), 'dashicons-smiley', 100 );

}

function wordFilterPage(){

?>

Hello world.

<?php

}

}

// put it in a variable so other plugins can use our reusable methods.

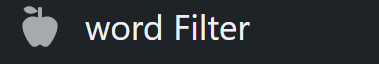
$ourWordFilterPlugin = new OurWordFilterPlugin;

Submenu

* add\_submenu\_page();
* add\_submenu\_page('parent menu (slug)', 'tab title', 'menu name', 'capability', 'added slug', 'html callback');

custom SVG

* ways
  + copy the SVG file content to the icon argument
    - change the binary format to asci (console dev tool)
      * btoa(``) // copy the result of this without quote after a comma.
    - 'data:image/svg+xml;base64, //WordPress will adjust the color



* + Give the url to the icon argument
    - plugin\_dir\_url(\_\_FILE\_\_) . 'custom.svg' // gives a URL to the directory that the current file lives in.
    - Be careful with its size

Html file

<div class="wrap">

<h1>Word Filter</h1>

<?php if($\_POST['justsubmitted'] == "true") echo 'Thank you'; ?>

<form method="POST">

<input type="hidden" name="justsubmitted" value="true">

<label for="plugin\_words\_to\_filter">

<p>Enter a <stron>comma-separated</stron> list of words to filter from your site's content.</p>

</label>

<div class="word-filter\_\_flex-container">

<textarea name="plugin\_words\_to\_filter" id="plugin\_words\_to\_filter" placeholder="bad, mean, awful, horrible"></textarea>

</div>

<input type="submit" name="submit" id="submit" class="button button-primary" value="Save Changes">

</form>

</div>

Add CSS file for this specific page

* add\_action('load-x'); // x is the hook name for the specific page that we are interenereste in.

function ourMenu(){

$mainPageHook = add\_menu\_page('Words To Filter', 'word Filter', 'manage\_options', 'ourwordfilter', array($this, 'wordFilterPage'), 'data:image/svg+xml;base64,1VjNIMTN … aIiAvPjwvc3ZnPg==', 100 );

add\_action("load-{$mainPageHook}", array($this, 'mainPageAssets'));

}

function mainPageAssets(){

wp\_enqueue\_style(' filterAdminCss', plugin\_dir\_url(\_\_FILE\_\_).'styles.css');

}

* wp\_enqueue\_style
* add\_menu\_page returns the hook name we are interested in.

Form Security

* Input hidden // in html of form start of the form
* $\_POST // php
* If we don't set the action for the form it will post to itself
* **update\_option('option','text area name')**
* class="updated" // Class is for WordPress, is the notification message

function handleForm(){

// save the text are to funciton

**update\_option**('plugin\_words\_to\_filter', $\_POST['plugin\_words\_to\_filter']);

?>

<!-- // class is for wordpress -->

<div class="updated">

Your filtered words were saved.

</div>

<?php

}

<textarea name="plugin\_words\_to\_filter" id="plugin\_words\_to\_filter" placeholder="bad, mean, awful, horrible"><?php echo esc\_textarea(get\_option('plugin\_words\_to\_filter')); ?></textarea>

* **Sanitizing** // You need to be careful with the post value coming.
  + **Sanitize\_text\_field()**

update\_option('plugin\_words\_to\_filter', **sanitize\_text\_field**($\_POST['plugin\_words\_to\_filter']));

* **Nonce** // cross-site request forgery. The user is the same user.
* **wp\_nonce\_field**('action name', 'the name of nonce value')
* inside the form:

<?php wp\_nonce\_field('saveFilterWords', 'ourNonce')?>

* Check the nonce when you want to save the value
* **wp\_verify\_nonce**('the value of set nonce', 'action name' );

function handleForm(){

if(wp\_verify\_nonce($\_POST['ourNonce'], 'saveFilterWords') AND current\_user\_can('manage\_options')){

// save the text are to funciton

update\_option('plugin\_words\_to\_filter', sanitize\_text\_field($\_POST['plugin\_words\_to\_filter']));

?>

<!-- // class is for wordpress -->

<div class="updated">

Your filtered words were saved.

</div>

<?php

}else { ?>

<div class="error">

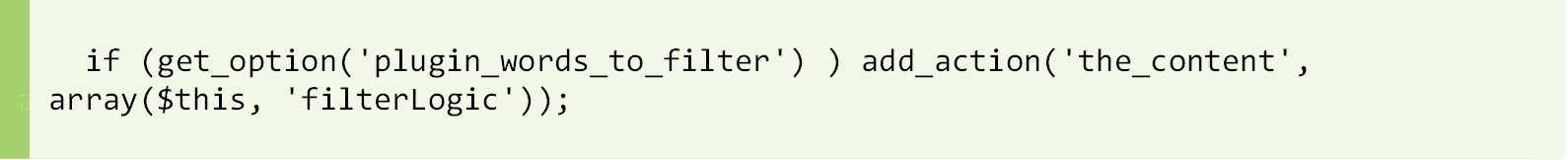
<p>Sorry, you do not have permission to perform that action.</p>

</div>

<?php

}

Filter the content



* explode(separate, string)
* **array\_map**(**'trim'**, array)
* **str\_replace**('array of words', replace text, string)

function **filterLogic**($content){

$words = get\_option('plugin\_words\_to\_filter');

$words = explode(',', $words);

$words = array\_map('trim', $words);

return str\_replace($words, esc\_html(get\_option('replacementText', ' \*\*\*')), $content);

}

* **settings\_errors**(); // add in the form with options.php. since we don't use add\_options\_page adds a submenu page to the Settings main menu which handles the settings itself we need it.

<form action="options.php" method="POST">

<?php

settings\_errors();

//group name

settings\_fields('replacementFields');

//slug name

do\_settings\_sections('word-filter-options');

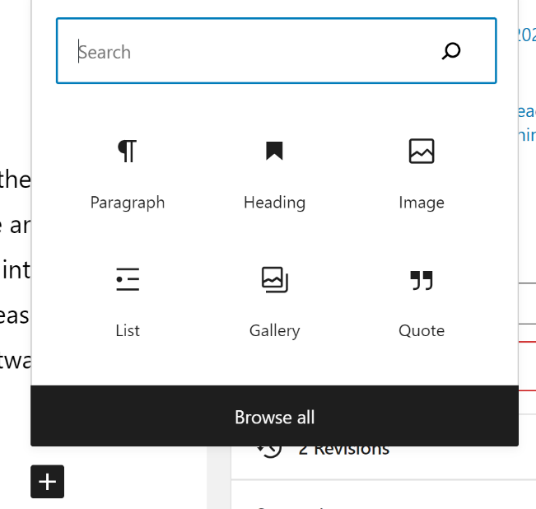
submit\_button();

?>

</form>

Custom Block type

* Gutenberg editor // block editor



Add the js resource in php

Static block

* The name of the plugin folder should not have spaces, we can use -
* **'enqueue\_block\_editor\_assets'** // add\_action hook
* **wp\_enqueue\_script**('name', address, list of dependencies);

if(!defined('ABSPATH')) exit;

class AreYouPayingAttention {

function \_\_construct()

{

//when on block screen run the js

add\_action('enqueue\_block\_editor\_assets', array($this, 'adminAssets'));

}

function adminAssets(){

wp\_enqueue\_script('ournewblocktype', plugin\_dir\_url(\_\_FILE\_\_). 'test.js', array('wp-blocks' ,'wp-element'));

}

}

$areYouPayingAttention = new AreYouPayingAttention();

js

* **wp.blocks.registerBlockType()** //WordPress adds the wp object to the global scope, it's the 'wp-blocks' we added to wp\_enqueue\_script to index.php. to make sure it is loaded before ou js file.
* **wp.blocks.registerBlockType('shortname', 'configuration object');**
* **wp.element.createElement('type of el', 'attributes', 'children/text', )** // can't return html in js, WordPress has a unique way.

wp.blocks.registerBlockType('ourlugin/are-you-paying-attention', {

title: "Are You Paying Attention",

icon: "smiley",

category: "common",

//admin screen

edit: function(){

return wp.element.createElement("h3", null, "This is from admin" );

},

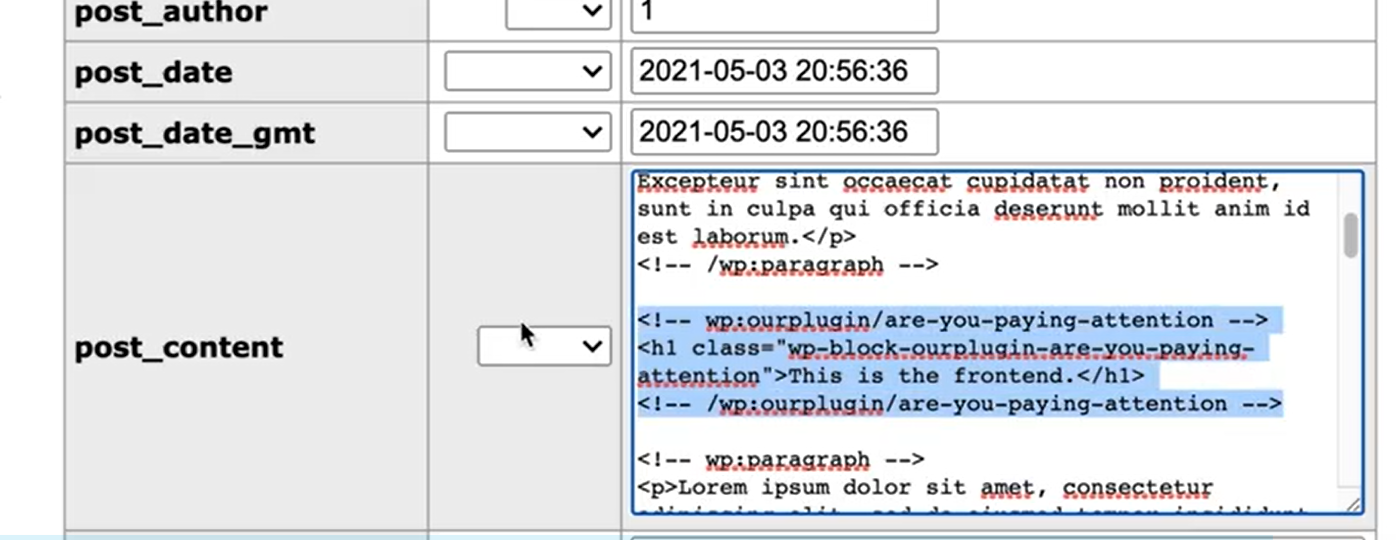
//public sees

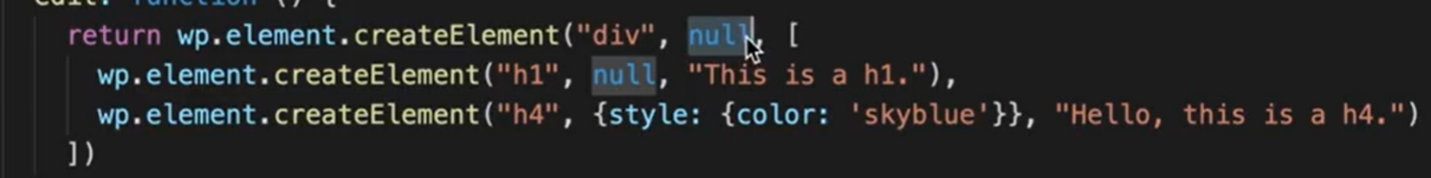
save: function(){

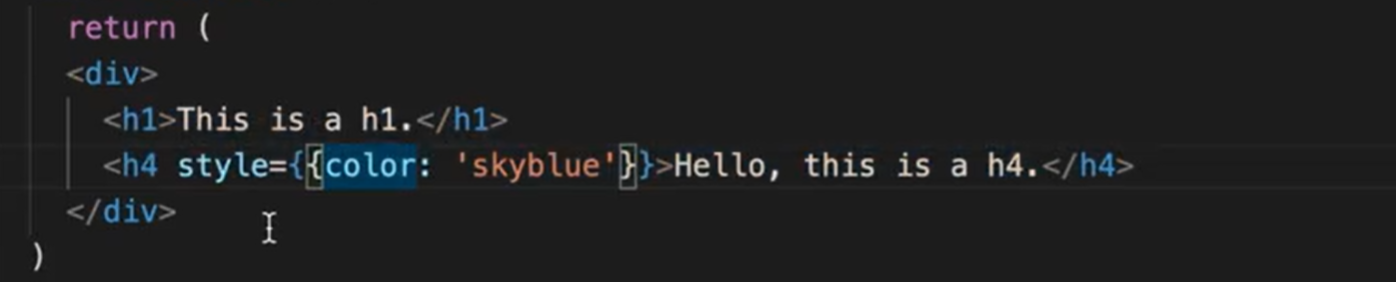
return wp.element.createElement("h1", null, "This is the frontend" );

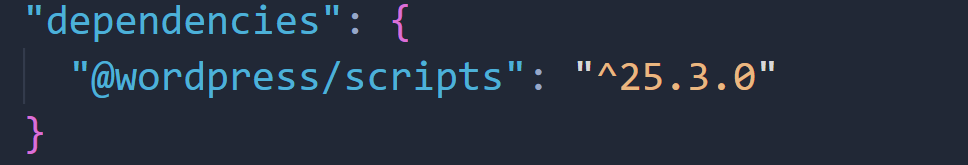
}

});

* The front end is saved in DB, but in the editor, the text is shown by js written.
* **JSX** // wp.element.createElement() is not practical for complex html
  + **Installation required**
  + **Script package**

Jsx

* in source file // we use jsx but it is converted ( with a tool in the middle) to the format
* Node js
* Npm
  + **npm init -y** // in the folder of the plugin. It creates a package.josn file to keep track of the dependencies.
  + **npm install @wordpress/scripts** (--save-dev // this options is not necessary anymore) // this package doesn't need extra configuration
    - npm ERR! Error: **ENOENT**
      1. clean cache: npm cache clean --force
      2. delete package.lock



* + create a **folder** named **src**
  + **index.js** // create this file, and move all the content we had to this file.
* To tell the package to convert it for us
  + In package.json in the script create these two tasks

"build": "wp-scripts build", // Build is onetime

"start": "wp-scripts start", // this will create automatically, good for developemetn

* **npm run start** // runs the above script
* Build // A folder is created with index.js with the converted code to wp-scripts
* **plugin\_dir\_url(\_\_FILE\_\_). 'build/index.js'** // change the address of wordpress wp\_enqueue\_script function.

JSX syntax

* Return () // () is not needed if we have only one line

"files.associations": {

"\*.js": "javascriptreact"

},

* In json setting file of vs code add: // To use html emmet (html tab trigger) in jsx
* <> </> // fragment tag, otherwise only one top-level tag is allowed
* /> //is needed if we don't write the closing tag
* {} // make it dynamic, write code

Condition ? '' : '' expression // in jsx we can't have if statement only expressions, something that boils down to a value if just controls the flow of the code, use the **ternary** operator

* { Condition && ( code ) } // multiple line if content

{isCorrectDelayed === true && index == props.correctAnswer && **(**

**<svg xmlns="http://www.w3.org/2000/svg" width="20" height="20" stroke="black" stroke-width="1" className="bi bi-check paying-attention-frontend--icon-correct" viewBox="0 0 16 16" >**

**<path d="M10.97 4.97a.75.75 0 0 1 1.07 1.05l-3.99 4.99a.75.75 0 0 1-1.08.02L4.324 8.384a.75.75 0 1 1 1.06-1.06l2.094 2.093 3.473-4.425a.267.267 0 0 1 .02-.022z"/>**

**</svg>)}**

Block type attribute

* Interactive. // input data so the user can type and edit
* It is the common ground that both edit and save can work on. It's how React framework works. Based on events update the values of the attributes (state).
  + Event listen // We don't work with PHP, html form that fires on submit
* Props.setAttributes WordPress sends the attributes to the edit and save functions and there in the edit function we use this function to update it on event change, in event callback we have access to the event object, to get the value.

wp.blocks.registerBlockType('ourlugin/are-you-paying-attention', {

title: "Are You Paying Attention",

icon: "smiley",

category: "common",

attributes: {

skyColor: {

type: "string"

},

grassColor: {

type: "string"

}},

//admin screen

edit: function(**props**){

function updateSkyColor(**e**){

props.setAttributes({skyColor:e.target.value})

}

function updateGrassColor(e){

props.setAttributes({grassColor:e.target.value})

}

return (

<div>

<input type = "text" placeholder="sky color" onChange = {**updateSkyColor**} />

<input type = "text" placeholder="grass color" onChange = {**updateGrassColor**}/>

</div>

)},

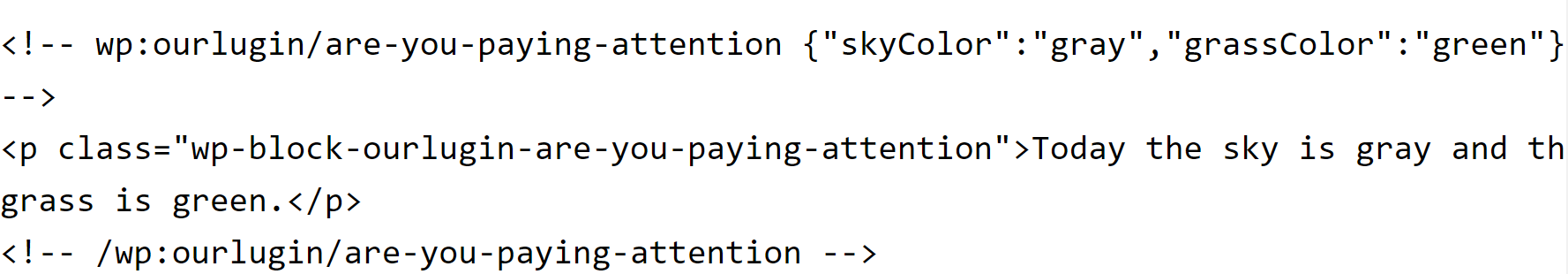
//public sees

save: function(props){

return (

<p>Today the sky is {props.attributes.skyColor} and the grass is {props.attributes.grassColor}.</p>

)}});

* In db in post content // it is stored in an html comment
  + Source // We can use this from WordPress to make it extract from the text part and don't save it in a comment.

attributes: {

skyColor: {

type: "string",

source: "text",

selector: ".skyColor"

},

grassColor: {

type: "string",

source: "text",

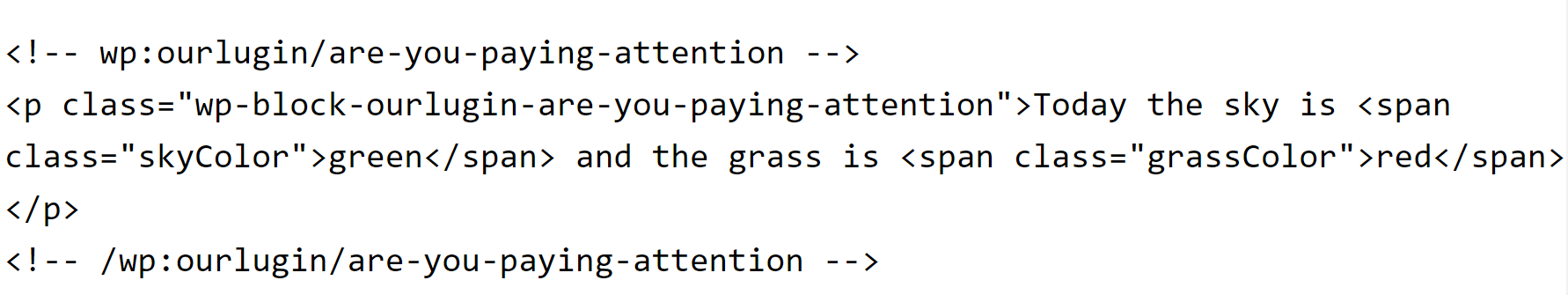
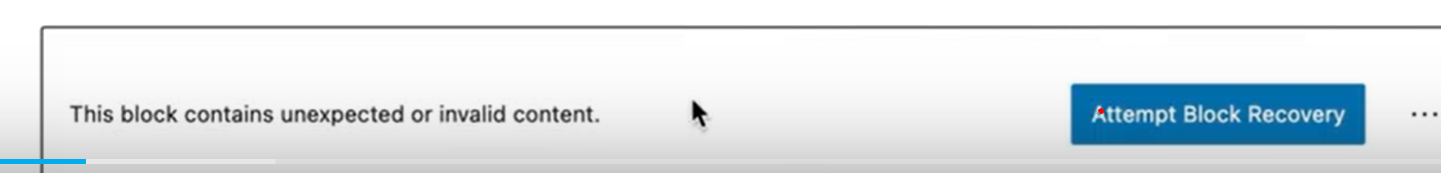
selector: ".grassColor"

}

},

* In save function

<p>Today the sky is **<span className="skyColor"> {**props.attributes.skyColor} </span> and the grass is **<span className ='grassColor'>** {props.attributes.grassColor} </span>.</p>

* The source of truce in the text
* if we make changes and we reload our post we face this error. Even though we don't use the source option, it still wp wants to make it available, therefore when you change something it can not trust the text and the values.
* To prevent it from happening
  + **deprecated** : [ ] // Keep a backup of the previous version
    - it's an array to accept every change in the array

save: function(props){

return (

<p>Today the sky is surprisingly {props.attributes.skyColor} and the grass is {props.attributes.grassColor}.</p>

)

},

// create a back up from previous version

**deprecated**: [**{**

attributes: {

skyColor: {

type: "string"

},

grassColor: {

type: "string"

}

},

save: function(props){

return (

<p>Today the sky is {props.attributes.skyColor} and the grass is {props.attributes.grassColor}.</p>

)

}

}]

* Static block type //The official wp documentation considers this the default way.
  + Problem: if the block type is changed you need to manually update the post, to see the changes in the front end, but that is an issue if you have too many posts.
* Dynamic block //
  + The save function returns null // since we moved the saved content from js to html callback
  + We move the responsibility from javascript to php, php returns the value
  + No static text is required to be saved in db, changes get an effect on the fly, because we get the text from php, not db.
  + Drawback: 2ms more for new php file // cache it

Dynamic Block

* Create a php with the exact name of namespace/name in registerBlockType in js and use the render function.
* **add\_ation('init', '')** // We don't create a blocktype in js we need a different action hook.

**wp\_register\_script()** // instead of enqueue, it won't be loaded we can reference the name when we want

* **register\_block\_type(**'namespace/blocktype', option array)
  + editor\_script // what you wrote in wp\_register\_script, which javascript to load
  + render\_callback // when we don't have the save callback in js
* **esc\_html()** // we can use this

class AreYouPayingAttention {

function \_\_construct()

{

//when on block screen run the js

add\_action('init', array($this, 'adminAssets'));

}

function adminAssets(){

wp\_register\_script('ournewblocktype', plugin\_dir\_url(\_\_FILE\_\_). 'build/index.js', array('wp-blocks','wp-element'));

register\_block\_type('ourplugin/are-you-paying-attention', array(

//which javascript to load

'editor\_script' => 'ournewblocktype',

'render\_callback' => array($this, 'theHTML')

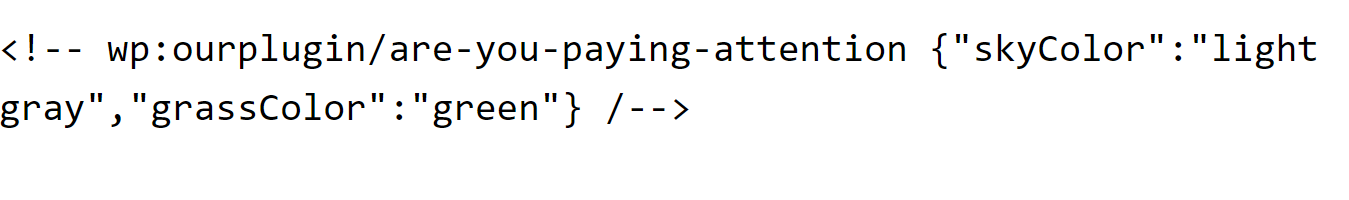
)); }

function theHTML($attributes){

// wp passes the attribute to this call back

return '<h1>Today the sky is '.$attributes['skyColor'].' and the grass is '.$attributes['grassColor'].'</1>';

}}



the clean way to include php var into a string

* **ob\_start()** // output buffer instead of concatenating for long html returned
* **ob\_get\_clean()**

function theHTML($attributes){

**ob\_start();** ?>

<h1>Today the sky is <?php echo $attributes['skyColor'] ?> and the grass is <?php echo $attributes['grassColor'] ?>!!!</1>

<?php return **ob\_get\_clean();**

}

Multiple answer question

Wp jsx components

* **<TextControl />** // need to be **imported**. Instead of writing our html we can use WordPress existing components
* When importing usually need to install with npm but, npm start run (to convert jsx) the automated solution that we wrote handles it and it sees it can find it within the global scope of WordPress. When they created this package they made sure when certain names show up, it converts it to a reference that looks in the browser global scope also with import react and reactDOM, it won't bundle up another copy of duplicate react, WordPress loads a copy of react for us. So the visitors don't need to download our copy too.
  + **'wp-editor'** // or we face an error in the post editor. It will look but we need to tell to load it in the global scope by writing a dependency

wp\_register\_script('ournewblocktype', plugin\_dir\_url(\_\_FILE\_\_). 'build/index.js', array(

'wp-blocks',

'wp-element',

**'wp-editor'**

));

Styles

* **Index.scss** // In the same folder with index.js create a file with css or scss with the same name.
* **import "./index.scss"** // Import the file from javascript and the automated system (npm run start) will extract it as a separate file and will include it in the build folder.
* **wp\_register\_style()** // Tell PHP to load the created css file in the build

wp\_register\_style('quizeditcss', plugin\_dir\_url(\_\_FILE\_\_). 'build/index.css');

* **'editor-style'** // in register\_block\_type, tell the block type to use it by name

**wp\_register\_style('quizeditcss', plugin\_dir\_url(\_\_FILE\_\_). 'build/index.css');**

wp\_register\_script('ournewblocktype', plugin\_dir\_url(\_\_FILE\_\_). 'build/index.js', array(

'wp-blocks',

'wp-element',

'wp-editor'

));

register\_block\_type('ourplugin/are-you-paying-attention', array(

//which javascript to load

'editor\_script' => 'ournewblocktype',

**'editor\_style' => 'quizeditcss',**

'render\_callback' => array($this, 'theHTML')

));

* Flex, FlexBlock, FlexItem, Button, Icon // import these components
  + FlexBlock // take space as much as it can
  + FlexItem // take the smallest amount of space it needs

import {TextControl, Flex, FlexBlock, FlexItem, Button, Icon} from "@wordpress/components"

* example

<TextControl label="Question: " />

<p>Answers:</p>

<Flex>

<FlexBlock>

<TextControl />

</FlexBlock>

<FlexItem>

<Button>

<Icon icon="star-empty" />

</Button>

</FlexItem>

<FlexItem>

<Button>

Delete

</Button>

</FlexItem>

</Flex>



* **isLink** // a prop that will remove the cursor pointer link effect on the padding.
* **isPrimary** // a prop for the button element that creates the button like wp blue buttons

.paying-attention-edit-block{

padding: 20px;

border-radius: 2px ;

-webkit-border-radius: 2px ;

-moz-border-radius: 2px ;

-ms-border-radius: 2px ;

-o-border-radius: 2px ;

border: 1px solid #d6d6d6;

background-color: #f1f1f1;

.mark-as-correct {

color: #ffd700;

position: relative;

top: -3px;

transition: transform 0.3s ease-out;

&:hover{

transform: scale(1.25) rotate(12deg) ;

-webkit-transform: scale(1.25) rotate(12deg) ;

-moz-transform: scale(1.25) rotate(12deg) ;

-ms-transform: scale(1.25) rotate(12deg) ;

-o-transform: scale(1.25) rotate(12deg) ;

}}

.attention-delete{

color: #ff0000;

position: relative;

top: -5px;

//& the top level rule that you are nested in: .attention-delete

&:hover{

color: #c20000 !important

}}}

<TextControl label="Question:" style={{fontSize: '20px'}} />

<p style={{ fontSize: '13px', margin: "20px 0 8px 0"}}>Answers:</p>

<Flex>

<FlexBlock>

<TextControl />

</FlexBlock>

<FlexItem>

<Button>

<Icon className ="mark-as-correct" icon="star-empty" />

</Button>

</FlexItem>

<FlexItem>

<Button isLink className ="attention-delete">

Delete

</Button>

</FlexItem>

</Flex>

<Button isPrimary >Add another answer</Button>

Adjust attributes for blocks

* **value = { props.attributes.question}** // a prop that updates values, where the first page loads pull the value from db, after each keystroke update the value in js ( not db )
* **onChange = {updateQuestion}** // updateQuestion is different compared to the grass and color update functions because we don't have the usual html but components. The textControl component returns the value.

function EditComponent (props){

function updateQuestion(value){

// update the js attribute in memory, it merges with it, the things we update

props.setAttributes({question: value})

}

return (

<div className = "paying-attention-edit-block">

{/\* the second {} is for the object with different properties \*/}

<TextControl label="Question:" value = {props.attributes.question} onChange = {updateQuestion} style={{fontSize: '20px'}} />

<p style={{ fontSize: '13px', margin: "20px 0 8px 0"}}>Answers:</p>

<Flex>

<FlexBlock>

<TextControl />

</FlexBlock>

<FlexItem>

<Button>

<Icon className ="mark-as-correct" icon="star-empty" />

</Button>

</FlexItem>

<FlexItem>

<Button isLink className ="attention-delete">

Delete

</Button>

</FlexItem>

</Flex>

<Button isPrimary >Add another answer</Button>

</div>

)

Array attribute

* array because of multiple answers, that we can loop through, by default we want at least one to appear in the admin. It's not an array of the JSX (store the JSX components). An array of the answer text string.

attributes: {

// multiple choice block

question: {

type: "string"

},

//

answers: {type: "array", default: [""]}

},

{/\* map for array js runs the function once fo each array \*/}

{props.attributes.answers.map((answer)=>{

// in JSX return () for multiple lines.

return (

<Flex>

<FlexBlock>

<TextControl value ={ answer } />

</FlexBlock>

<FlexItem>

<Button>

<Icon className ="mark-as-correct" icon="star-empty" />

</Button>

</FlexItem>

<FlexItem>

<Button isLink className ="attention-delete">

Delete

</Button>

</FlexItem>

</Flex>

)

})}

Changeable input after loaded from the state (attribute)

* **onChange** // You may notice after loading the page with values in input, the input does not allow typing. This is because react input has a dynamic value that gets it from the state (attribute). The dom element is not the source of truth. They need the onChange prop. A function that manipulates the state data.
* **props.attributes.answer.push(newValue**) // This is not correct. in React we don't change the value of the state directly. We give the new value, and it handles the update.
  + **concat([])** // Create a new copy that can be altered. Concat with an empty array and it will return a new copy.

<FlexBlock>

<TextControl value ={ answer } onChange = { (newValue, prevValue) => {

const newAnswers = **props.attributes.answers.concat([])**

newAnswers[index] = newValue

props.setAttributes({answers: newAnswers })

}} />

</FlexBlock>

Add new element to an array attribute

* When the attribute is updated, it will run the map function that is set on the attribute and is responsible to show the input html again and produce a new component for it. But we can't use push, we can use concat([""]).

<Button isPrimary onClick={()=>{

props.setAttributes({answers: props.attributes.answers.concat([""]) })

}}>Add another answer</Button>

Delete an element from an array attribute

* assign to onClick an anonymous function // To be able to pass the argument index

<Button isLink className ="attention-delete" onClick={() => deleteAnswer(index)} >

* Filter(item, index) // excludes the items based on the returned condition.

const newAnswers = props.attributes.answers.filter((item, index)=>{

return index != indexToDelete

})

props.setAttributes({answers: newAnswers })

Type number of attribute

* **default: undefined** // If the number is the index 0, when we want to check if the user has selected the correct answer, it will be a problem as zero will be evaluated as false.
* **onClick** // to button
* **expression** // in jsx we can't have if statement, ternary

<Button onClick = {() => markAsCorrect(index)}>

<Icon className ="mark-as-correct" icon={(props.attributes.correctAnswer == index ) ? "star-filled" : "star-empty"} />

</Button>

function markAsCorrect(index){

// to toggle

if(props.attributes.correctAnswer == index){

props.setAttributes({correctAnswer: undefined })

}else{

props.setAttributes({correctAnswer: index })

}

}

* if the **correct answer** is deleted

if(indexToDelete == props.attributes.correctAnswer){

props.setAttributes({correctAnswer: undefined })

}

* block editor as a whole, multiple quizzes// disable the save button if the correct answer is not selected, all instances of the block should have a correct answer.
  + outside of EditComponent
* **wp.data.select("core/block-editor").getBlocks() //** In js console, returns all the blocks for the current post as an array of objects
* **wp.data.subscribe(function(){})** // it will run the function anytime any data of the block as a whole change, always fresh data
* **wp.data.dispatch**("type of action we want to dispatch").lockPostSaving('name of criteria') // to lock the save action

// immediately invoked function expression(iife), to avoid unique names

// variables scope only limited to the function

(function (){

let locked = false;

wp.data.subscribe(function(){

const results = wp.data.select("core/block-editor").getBlocks().filter(function(block){

return block.name == "ourplugin/are-you-paying-attention" && block.attributes.correctAnswer == undefined

})

if(results.length && locked == false){

locked = true

wp.data.dispatch("core/editor").lockPostSaving("noanswer")

}

if(!results.length && locked ){

locked = false

wp.data.dispatch("core/editor").unlockPostSaving("noanswer")

}

})

})()

block front end

* **wp-scripts start** // knows src/index.js, not other user-created files
* **list the files**

"build": "wp-scripts build src/index.js scr/frontend.js",

"start": "wp-scripts start src/index.js scr/frontend.js",

* If You're Using a Block Theme: Load Your JS In The Footer
* frontend.scss // import in js, load it in php. Two ways:
  + register\_block\_type // in the option array
    - "script" => to an asset for frontend // This will **always** **load the asset** even if we don't have the block in our post (frontend)
    - "style" => asset
  + In **theHTML callback** of the **register\_block\_type in index.php**, since callback only **is called if the block type is used**.
    - Wp\_enqueue\_script() // wp loads the wp\_enqueue\_script file with the same short name only once, even if we have multiple instances of the same block in the page. That's good.

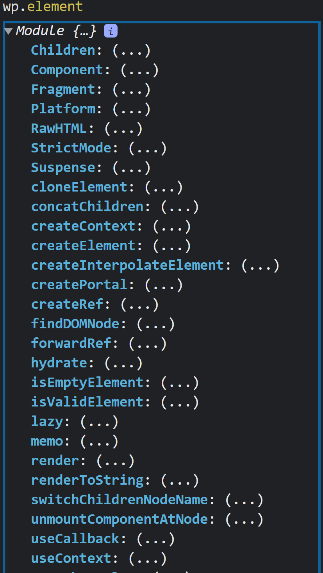
if(!is\_admin()){ // not to load in admin

wp\_enqueue\_script('attentionFrontend', plugin\_dir\_url(\_\_FILE\_\_) . 'build/frontend.css', array('wp-element'));

wp\_enqueue\_style('attentionFrontendStyles', plugin\_dir\_url(\_\_FILE\_\_) . 'build/frontend.css');

}

React dom

* wp-element // is the WordPress version of react, they abstracted react into their own script. We don't need to install react because we defined the dependency by array('wp-element') in wp\_enqueue\_script.
* **wp.element** // In js console will show that it is React component with its core function but has react-dom built into it (render method is there). It won't download them in its webpack config looks in the browser's global scope for wp-element. Based on js build file size we can see it doesn't load the react onto it. All plugins will use only one react.

**import React from 'react'**

**import ReactDOM from 'react-dom'**

import "./frontend.scss"

// select all instances of that block type. don't forget the dot!

const divsToUpdate = document.querySelectorAll(".paying-attention-update-me")

//foreach create a component

divsToUpdate**.forEach**(function(div){

**ReactDOM.render(<Quiz />, div)**

//to run the code on the fly, if new posts are added by lazy loading, what elements have already been hydrated with javascript

// so we don't update them again if new ones are added

div.classList.remove("paying-attention-update-me")

})

// JSX component

function Quiz(){

**return (**

**<div className="paying-attention-frontend">**

**Hello from React**

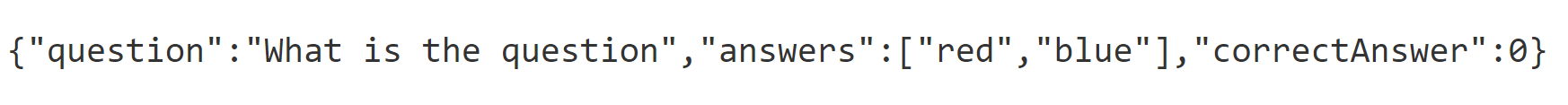
**</div>**

**)**

}

Block data from php to js

* $attribute is stored in json syntax, in db, then is sent to register\_block\_type callback as a php array, then we convert it back to josn for the front end.
* But we need to pass the data to our new front-end js. No official wordpress way to do this.
* How to use js in react to render your block types in the front end? Many ways.
* **wp\_json\_encode($attribute)** // echo this in the div we set after ob\_start() in index.php. It converts the array to text with **angle quotes** that do not work with **json interpreter** on the js, and we have to remove them. Wrap it in **a pre-tag.**



ob\_start(); ?>

<div class="paying-attention-update-me"><pre><?php echo wp\_json\_encode($attributes) ?></pre></div>

<?php return ob\_get\_clean();

}

* JSON.parse(text) // text to dynamic json
* **{…data} destructuring for prop** // Efficient way to pass the data to the component. It will return each item of json as separate props. So you don't need nested props.data.question just props.question.

Onclick js

* Getbootstrap.com // icon tab, free icons, svg code => copy html (svg)
* In react we don't look for the element with the correct message, we can't mutate the dom directly. It can track its own internal state. And our JSX, can **react** to that app state.
  + isCorrect // a state
  + useState // describes the current state of our component, we don't directly mutate it, we give it a new value, and react handles keep track of it. Whenever the state changes, react will rerender our function Quiz. Whenever the state changes it well knows we want to render the Quiz component to the given dom, it renders it again when the state changes.
  + react **useState()** will return an array, the first **easy access to state**, second **to change it**.

function Quiz(props){

const [isCorrect, setIsCorrect] = **useState(undefined)** //default value

function handleAnswer(index){

if(props.correctAnswer == index){

setIsCorrect(true)

// document.querySelector('.correct-message').classList.add('correct-message\_\_visible')

}else{

setIsCorrect(true)

// document.querySelector('.incorrect-message').classList.add('incorrect-message\_\_visible')

} }

retrn()}

* + undefined is the **default value** for this state we defined.

<div className={"correct-message " + ( isCorrect === true ? "correct-message--visible" : "" )} >

* **Iscorrect** will not fire again on the same answer (not on incorrect ones if the previously clicked one was incorrect or the correct one if the prev clicked one was correct) because the state is false and there is no change for React to react to. The class on Div is there and won't get changed. Still you

function handleAnswer(index){

if(props.correctAnswer == index){

setIsCorrect(true)

// setIsCorrect((prevState) => true);

// setIsCorrect(true)

// setTimeout(() => {

// setIsCorrect()

// }, 2000);

// document.querySelector('.correct-message').classList.add('correct-message\_\_visible')

}else{

if(isCorrect == false) { setIsCorrect((prevState) => {isCorrect: undefined}) }

setIsCorrect(false)

// setTimeout(() => {

// setIsCorrect(undefined)

// }, 2000);

// document.querySelector('.incorrect-message').classList.add('incorrect-message\_\_visible')

}

}

React.useEffect(() => {

alert(isCorrect)

if(isCorrect === true || isCorrect === false){

setIsCorrect(undefined)

}

}, [isCorrect]);

Change state multiple times, do things after state changes

* **useEffect**(**function** when we want to run, an **array** of when we want to run) // a hook. the second arg, what properties or piece of state are we watching, watch changes, an array. If the [] is empty, it runs only once when the component renders. We can return a function which is called the cleaner function, and call it when the component is deleted or unmounted.

React.useEffect(() => {

// alert(isCorrect) // order of alert: undefined - true/false - undefined

if(isCorrect === false){

setTimeout(() => {

setIsCorrect(undefined)

}, 2600);

}

}, [isCorrect]);

* Disable clicking after true

**onClick**={isCorrect === true ? () => handleAnswer(index): ''}

* Avoid multiple animations to be shown, distracting, don't change the style of the answer right after changing state and the correct message
  + Wait half of the animation for the correct message
  + Use another state to show that // to track in between timers, and use on the dives for the change of style

React.useEffect(() => {

// alert(isCorrect) // order of alert: undefined - true/false - undefined

if(isCorrect === false){

setTimeout(() => {

setIsCorrect(undefined)

}, 1400); }

// to animate the final change on answers

**if(isCorrect === true){**

**setTimeout(() => {**

**setIsCorrectDelayed(true) // use another state for this**

**}, 600);** }

}, [isCorrect]);

* the && says only when before is true include it since we can't set an if in jsx

return (

// **the parenthesis in className is required**

<li className = {(isCorrectDelayed === true && index == props.correctAnswer ? "no-click" : "") + (isCorrectDelayed === true && index != props.correctAnswer ? "fade-incorrect" : "")} onClick={isCorrect === true ? '' : () => handleAnswer(index) }>

{isCorrectDelayed === true && index == props.correctAnswer && (

<svg xmlns="http://www.w3.org/2000/svg" width="20" height="20" stroke="black" stroke-width="1" className="bi bi-check paying-attention-frontend--icon-correct" viewBox="0 0 16 16" >

<path d="M10.97 4.97a.75.75 0 0 1 1.07 1.05l-3.99 4.99a.75.75 0 0 1-1.08.02L4.324 8.384a.75.75 0 1 1 1.06-1.06l2.094 2.093 3.473-4.425a.267.267 0 0 1 .02-.022z"/>

</svg>)}

{isCorrectDelayed === true && index != props.correctAnswer && (

<svg xmlns="http://www.w3.org/2000/svg" width="16" height="16" stroke="black" stroke-width="1" className="bi bi-x paying-attention-frontend--icon-wrong" viewBox="0 0 16 16">

<path d="M4.646 4.646a.5.5 0 0 1 .708 0L8 7.293l2.646-2.647a.5.5 0 0 1 .708.708L8.707 8l2.647 2.646a.5.5 0 0 1-.708.708L8 8.707l-2.646 2.647a.5.5 0 0 1-.708-.708L7.293 8 4.646 5.354a.5.5 0 0 1 0-.708z"/>

</svg>

)}

{answer}

</li>)})}

Custom options in the admin

Righthand options area

* import {**InspectorControls**} from "wordpress/block-editor" // in index.js for admin
  + "@wordpress/block-editor"

import {InspectorControls} from **"@wordpress/block-editor"**

* Also, add

import {TextControl, Flex, FlexBlock, FlexItem, Button, Icon, PanelBody, PanelRow} from "@wordpress/components"

* in return for editComponent callback // WordPress knows what to do with this it won't include it in the main area

return (

<div className = "paying-attention-edit-block">

<InspectorControls>

<PanelBody title="Background Color">

<PanelRow>

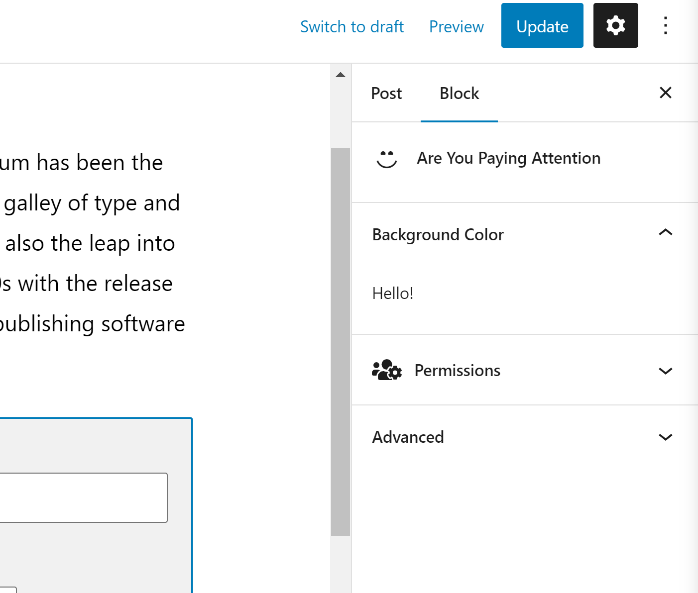
Hello!

</PanelRow>

</PanelBody>

</InspectorControls>

….



Color picker component

* **ColorPicker** // from @wordpress/components
  + color // an attribute
  + onChangeComplete // an attribute

import {TextControl, Flex, FlexBlock, FlexItem, Button, Icon, PanelBody, PanelRow, ColorPicker} from "@wordpress/components"

<InspectorControls>

<PanelBody title="Background Color">

<PanelRow>

<ColorPicker />

</PanelRow>

</PanelBody>

</InspectorControls>

* Add an attribute to save it in db
  + props.setAttributes

bgColor: {type: "string", default: "#EBEBEB"}

<ColorPicker color={props.attributes.bgColor} onChangeComplete ={x => props.setAttributes({bgColor: x.hex})} />

* To change the color on the fly

<div className = "paying-attention-edit-block" style={{backgroundColor: props.attributes.bgColor}}>

* React Color // Third community react packages for colorPicker. It's not related to WordPress

import {ChromePicker} from "react-color"

<ChromePicker color={props.attributes.bgColor} onChangeComplete ={x => props.setAttributes({bgColor: x.hex})} />

Inline toolbar, Text alignment option

* Index.php
  + **BlockControls** // inline toolbar
  + **AlignmentToolbar**

import {InspectorControls, **BlockControls**, **AlignmentToolbar**} from "@wordpress/block-editor"

* Add attribute

<BlockControls>

<AlignmentToolbar value={props.attributes.theAlignment} onChange={x => props.setAttributes({theAlignment:x})} />

</BlockControls>

theAlignment:{type:"string", default: "left"}

* Front-end

style={{backgroundColor: props.bgColor, textAlign: props.theAlignment}}

* Set Preview for block type.
  + example property of registerBlockType

example: {

attributes: { // multiple choice block

question: "What is my name",

answers: ['Meowsalot', 'Barksalot', 'Purrsalot'],

correctAnswer: 3,

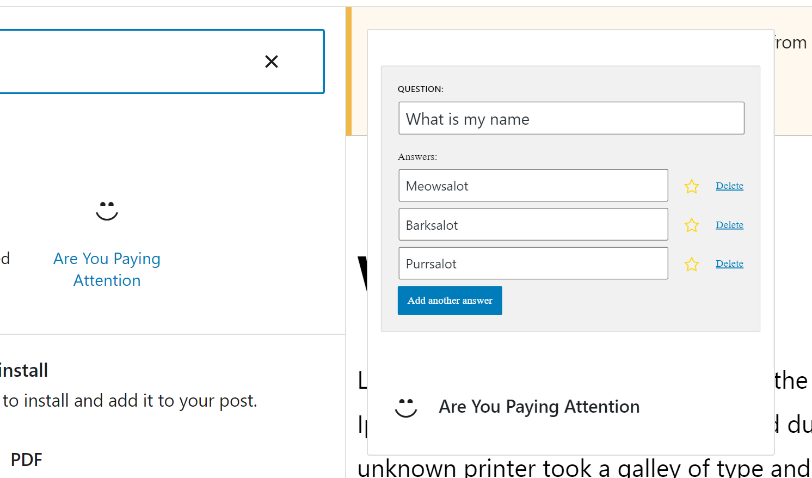
bgColor: 'center',

theAlignment:'#9ED3C8'

}

},

* + description: 'This is a quiz block'



block.json

* in the root of the plugin folder
* You can define the title, attributes,
* "$schema" // which property should and should not be contained
* Define the address to the asset // then no need for manually enqueueing them
* In block.json

{ // the similar ones you can delete in index title for example, attributes ...

// which property this file should and should not contain

"$schema": "https://schemas.wp.org/trunk/block.json",

"apiVersion": 2,

"name": "ourplugin/are-you-paying-attention",

// for humans

"title": "Are You Paying Attention",

// admin editor screen

"editorScript": "file:./build/index.js",

"editorStyle": "file:./build/index.css",

"script": "file:./build/script.js",

"viewScript": [ "file:./build/view.js", "example-shared-view-script" ],

"style": [ "file:./build/style.css", "example-shared-style" ]

}

* register\_block\_type // in php file the first argument of this should be addressed to the root of where block.json is

public function adminAssets()

{

// do not need this because of block.json

// wp\_register\_style('quizeditcss', plugin\_dir\_url(\_\_FILE\_\_). 'build/index.css');

// wp\_register\_script('ournewblocktype', plugin\_dir\_url(\_\_FILE\_\_). 'build/index.js', array(

// 'wp-blocks',

// 'wp-element',

// 'wp-editor'

// ));

register\_block\_type**(\_\_DIR\_\_,** array(

// which javascript to load, no longer need because of block.json

// 'editor\_script' => 'ournewblocktype',

// 'editor\_style' => 'quizeditcss',

**'render\_callback' => array($this, 'theHTML')**

));

}

Some issues with the appearance in Zoom and the right-site menu with this method

* "apiVersion": 2, // in block.json, in the new version we have to micro-manage the block type ourselves. In the other version, wordpress will add the wrap div around the block, which will enable clicking and right-site menu
* **useBlockProps**

import {InspectorControls, BlockControls, AlignmentToolbar, **useBlockProps**} from "@wordpress/block-editor"

* in edit property of index.js

function EditComponent (props){

const blockProps = useBlockProps() …

return (

// our wrapper div use spread syntax, all properties of blockProps will be applied here

        <div {...blockProps} >

…

* Class/Style props // It also includes the class and style so we have to add these props to the useBlockProps

const blockProps = useBlockProps({

className: "paying-attention-edit-block",

style: {backgroundColor: props.attributes.bgColor}

});

Front-end

* **"script" – "style"** // in block.json for **both frontend and backend**
* **No viewStyle!** //
  + Enqueue script yourself
  + Don't create selectors that are similar in front and admin
* **"viewScript"** //is only for the front-end but wordpress **will not load** if you use php **render callback** function in **register\_block\_type()**
  + Use the "script" that is for both // by default include it in the header, not the footer, no dom
    - Document.addEventListener("DOMContentLoaded", function()) // Wrap everything in frontend.js except imports into it

A select block type with post data

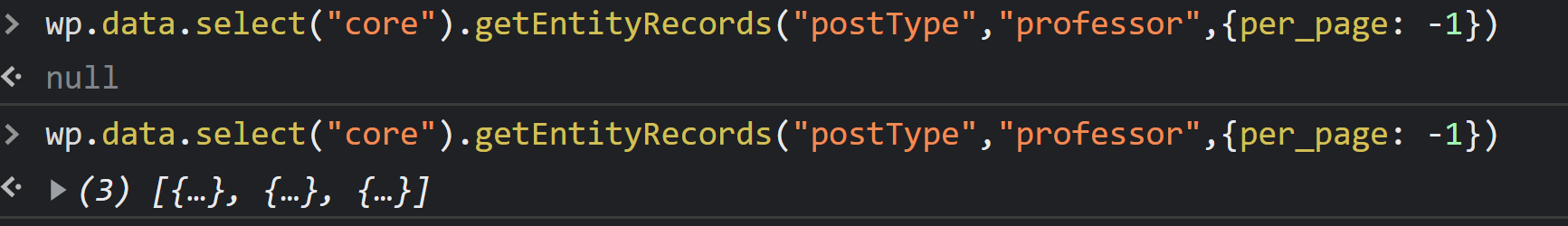
* In the index js

<select onChange={(e)=>props.setAttributes({profId: e.target.value})}>

<option value="">Select a professor</option>

</select>

* How to load data
  + Send Request to rest API
  + wp.data.select("core").getEntityRecords("postType","professor",options) // no need for HTTP request, wp has set tools so we can load data. it returns null if we try it in the console because it's asynchronous



wp.data.select("core").getEntityRecords("postType","professor",{per\_page: -1})

null

* to deal with the asynchronous problem null that requires time to fetch the data, we have a react solution by wp.

Import the tools

* use select will run our function (arrow function) again whenever the data store changes, the first time returning null, the second time the data.

import {useSelect} from "@wordpress/data"

const allProfs = useSelect(select=>{

//select is the tool

return select("core").getEntityRecords("postType","professor",{per\_page: -1})

})

Loading icon while waiting

* Don't load anything when not ready

if(allProfs == undefined) return <p>Loading...</p>

<select onChange={(e)=>props.setAttributes({profId: e.target.value})}>

<option value="">Select a professor</option>

{/\* map should have return \*/}

{allProfs.map( prof =>

{

return (

<option value={prof.id} selected={props.attributes.profId == prof.id}>

{prof.title.rendered}

</option>

)

}

)}

</select>

Another file for front-end html

* A function in index.php (this file can have any name)

function renderCallback($attributes){

if($attributes['profId']){

wp\_enqueue\_style('featuredProfessorStyle');

return generateProfessorHTML($attributes['id']);

}else{

return NULL;

}

}

* in index.php

require\_once plugin\_dir\_path(\_\_FILE\_\_) . 'inc/generateProfessorHTML.php';

* In the new file

<?php

function generateProfessorHTML($id){

$profPost = new WP\_Query(array(

// we put this so for any suspicious give id we only look in profs

'post\_type' => 'professor',

'p' => $id

));

while($profPost->have\_posts()){

$profPost->the\_post();

//since our return is a lot of html

ob\_start(); ?>

<div class="professor-callout">

<div class="professor-callout\_\_photo"></div>

<div class="professor-callout\_\_text">

<h5><?php the\_title() ?></h5>

<p><?php echo wp\_trim\_words(get\_the\_content(), 30); ?></p>

</div>

</div>

<?php

wp\_reset\_postdata();

return ob\_get\_clean();

}

}

?>

* **get\_field** // Related programs
  + **array\_key\_last** // condition

<div class="professor-callout\_\_text">

<h5><?php the\_title() ?></h5>

<p><?php echo wp\_trim\_words(get\_the\_content(), 30); ?></p>

<?php

$relatedPrograms = get\_field('related\_programs');

if($relatedPrograms){

?>

// or **wp\_stip\_all\_tags**

<p><?php **esc\_html**(get\_the\_title()?> teaches:

<?php

// print\_r($relatedPrograms);

// 0 => object

foreach($relatedPrograms as $key => $program){

echo get\_the\_title($program);

if($key != array\_key\_last($relatedPrograms) && count($relatedPrograms) >1){

echo ', ';

}}

?>.

</p>

<?php

}

?>

<p><strong><a href="<?php the\_permalink() ?>">Learn more about <?php the\_title() ?> &raquo;</a></strong></p>

</div>

Load the data in the backend and server-side render html practice

* Many ways to do this. Pros and Cons.
* Since we want server-side rendered html for the public front-end for SEO and accessibility reasons, and to avoid duplicating our code.
  + **Frontity** // react framework for wordpress. In the usual Wordpress to create a template there is a way to define it in jsx (for client-side) and use it also for serverside html.
  + But block types are different. // Most affordable web hosts will not allow running node.js alongside the WordPress app, so on an affordable web host, no way to have jsx use on the **server side**!!
  + So we use php render html in the edit block. How?
    - Create a new field for professor data for HTML// but for each profs we have this, and what if we have a lot of profs?
    - Create a new **rest api endpoint for the html**!!

function \_\_construct()

{

add\_action('init', [$this, 'onInit']);

add\_action('rest\_api\_init',[$this, 'profHTML']);

}

function profHTML(){

register\_rest\_route('featuredProfessor/v1','getHTML', array(

'methods' => WP\_REST\_SERVER::READABLE,

'callback' => [$this, 'getProfHTML']

));

}

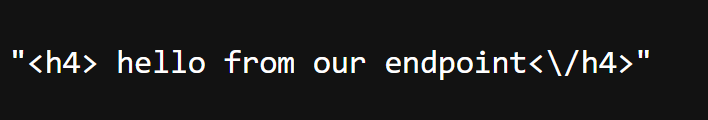
function getProfHTML(){

// return json, single string of text is valid json

return '<h4> hello from our endpoint</h4>';

}

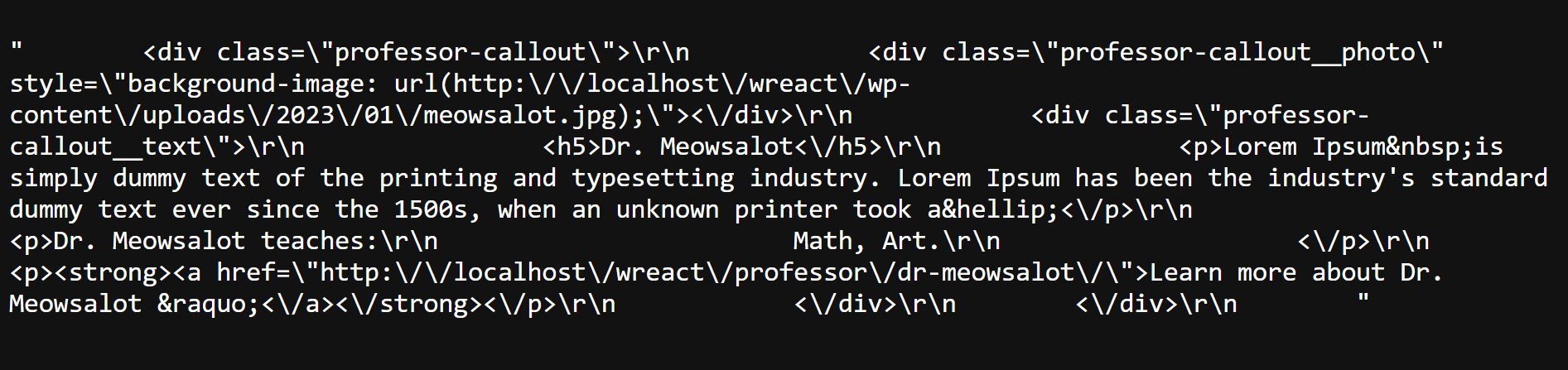
* Backslash is for json to help with storing the html content



New endpoint for one specific professor (id)

* To use the existing html function for the professor, we need the id of prof.
* $data // access the url variables, set it in the url of the new endpoint

http://localhost:3000/wreact/wp-json/featuredProfessor/v1/getHTML?profId=43



Fetchapi and state to use endpoint within react component

* State // to keep track of html prev value, for asynchronous and dynamic access to the endpoint, so react can react to change and show it to us. In edit component.

const[thPreview, setThePreview] = useState("")

* The **event** to react to
  + When the value of **id changes**.
  + **useEffect** // A nice way to do something when a specific piece of data changes
* **Async function** // inside the EditComponent

const[thePreview, setThePreview] = useState("")

useEffect(()=>{

// with this if, it won't show the last professor added when select is undefined

if(props.attributes.profId){

//fetch the new version of

// no need to use fetch manually, it will parse the json itself too

// since it returns a promise you can use await,

// use effect can't accept an async function directly (before () of arrow function), but defining an asynch function inside it and the call it.

**async function go()**{

const response = **await** **apiFetch**(({

// **does not** need the **rest of address**.

path: `/featuredProfessor/v1/getHTML?profId=${props.attributes.profId}`,

method: "GET"

}))

**setThePreview(response)**

}

**go()**

**}**

}, [props.attributes.profId])

* **dangerouslySetInnerHTML = {{\_\_html: thePreview}}** // <div>{thePreview}</div> can not be used, react shows only the text instead of rendering it and displaying it as real html. For safety.
  + That's the drawback of using a PHP html template!

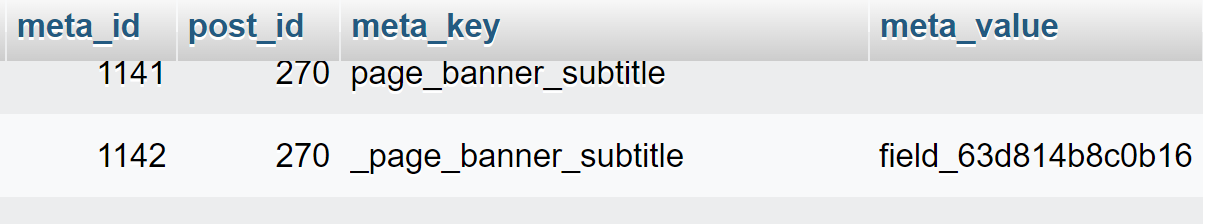
<div dangerouslySetInnerHTML ={{\_\_html: thePreview}}></div>

Post Meta Data, to create relationships with profs and featured profs

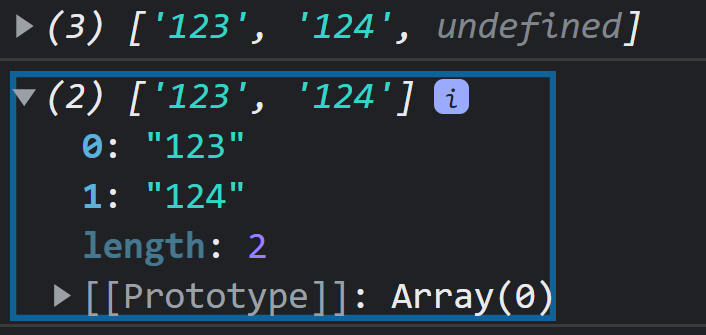
* How to access the id stored in the post block comment on prof page?

Create a new post meta

* in the postmeta table, Set the post\_id to the blog post featuring the profs, and meta\_value the actual profs id.



* In block index.js, we have an event with useEffect on id-attribute change, use it to update the metadata
* **wp.data.dispatch("core/editor").editPost({meta: {featuredprofessor: profsForMeta }})** // Wordpress javascript tool that is available in global scope. Only saves the data in the datastore of javascript memory.
* We can't set props.attributes.profId for the meta value since we may not have one instance of the block type. we don't want them to override each other.
  + Look at it as a whole
* filter((x, index, arr)) // the array is the array we are filtering itself, to filter duplicates, return true if it does not already exist.
  + **indexOf**



function updateTheMeta(){

// select all block types and filter for our block type

// filter(x => x.name == 'ourplugin/featured-professor') returns all sort of property of that blocktype

// map is used when we want to do something for each value of array and return an array, like only ids

// filter again no need to store the same id twice if we have the same featured block on a blog post!!

const profsForMeta =wp.data.select("core/block-editor")

.getBlocks()

.filter(x => x.name == 'ourplugin/featured-professor')

.map(x=>x.attributes.profId)

.filter((x, index, arr) =>{

// indexOf will return the first one. If we have it a second time the index don't match

return arr.indexOf(x) == index

})

console.log(profsForMeta)

wp.data.dispatch("core/editor").editPost({meta: {featuredprofessor: profsForMeta }})

}

Registering meta to save

* register custom metadata and then click update button
* **register\_meta**('type of metadata: post, comment, user', 'name of meta, it should match in js', array('options')). Add to init callback of index.php.
* **'single' => false** // wp tries to store an array for the meta\_value, may be serialized and database lookup performance will be slower, it will create three rows in the table **for each profs id** of block type in a blog post. Post-id and meta\_key will be the same, and meta\_value will be different.

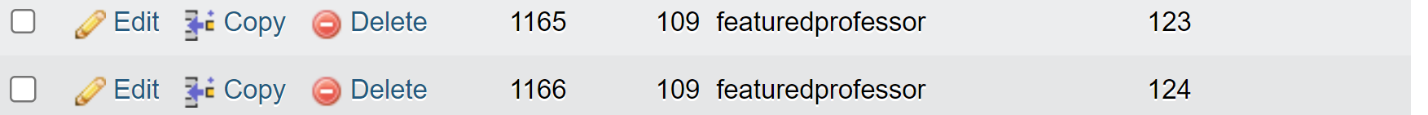
register\_meta('post', 'featuredprofessor', array(

'show\_in\_rest' => true,

'type' => 'number',

'single' => false

));



Delete the meta if block type is deleted

* Event on mounting (getting deleted)
* useEffect on the empty array and cleanup function
* Save the post

useEffect(()=>{

// cleaner function, fresh copy of Meta

return () => { updateTheMeta() }

}, [])

Use meta in a post

* Include everything about the plugin in the plugin
* Add\_filter // To filter the content of the professor, in index.php

class FeaturedProfessor

{

function \_\_construct()

{

add\_action('init', [$this, 'onInit']);

add\_action('rest\_api\_init',[$this, 'profHTML']);

**add\_filter('the\_content', [$this, 'addRelatedPosts']);**

}

**// add the posts that featured profs in**

**function addRelatedPosts($content){**

**if(is\_singular('professor') && in\_the\_loop() && is\_main\_query()){**

**// function in a different file and require it!**

**//has access the to the current id;**

**return $content . relatedPostsHTML(get\_the\_id());**

**}**

**return $content;**

**}**

* In relatedPostsHTML.php
  + A query on posts with meta\_query, which will look for metadata too.

<?php

function relatedPostsHTML**($id**){

// wp\_query will also query the meta table

$postsAboutThisProf = new WP\_Query(array(

'posts\_per\_page' => -1,

'post\_type' => 'post',

'meta\_query' => array(

**array(**

**'key' => 'featuredprofessor',**

**'compare' => '=',**

**'value' => $id**

**)**

)

));

ob\_start();

if($postsAboutThisProf -> found\_posts){ ?>

<p><?php the\_title()?> is mentioned in the following posts:</p>

<ul>

<?php

while($postsAboutThisProf->have\_posts()){

$postsAboutThisProf->the\_post();

?>

<li><a href="<?php the\_permalink(); ?>"><?php the\_title(); ?></a></li>

<?php

}

?>

</ul>

<?php

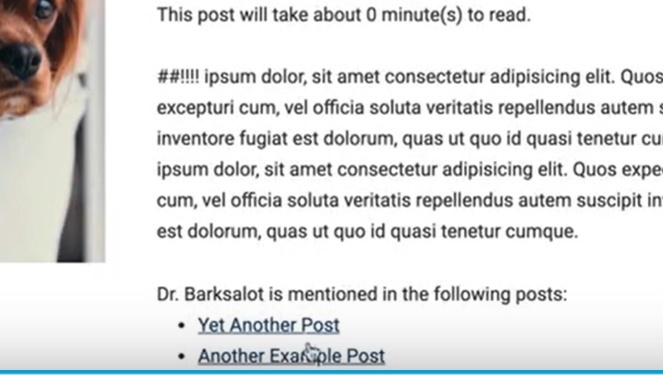
}

wp\_reset\_postdata();

return ob\_get\_clean();

}

?>



Localization/translation in JS

* In php we had loco translate
* In js we still need php to enable translate, in index.php in the comment above the file:

Text Domain: featured-professor

Domain Path: /languages

* Create a folder /languages. In the init callback of index.php for the plugin

// support translation for this plugin

load\_plugin\_textdomain('featured-professor',false, dirname(plugin\_basename(\_\_FILE\_\_)) . '/languages');

* **wp\_set\_script\_translations(**'handle name of script', 'text-domain', 'address languages folder'); // Tie the javascript file to the translation system. Add it in the init hook callback.

wp\_set\_script\_translations('featuredProfessorScript', 'featured-professor', plugin\_dir\_path(\_\_FILE\_\_) . '/languages');

In js

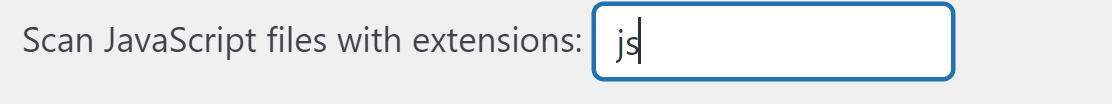
* wp.i18n.\_\_ // shortcut from the global scope of WordPress instead of importing since it may have some issues, instead of typing wp.i18n.\_\_ , type \_\_

const \_\_ = wp.i18n.\_\_

* in jsx html

{\_\_("Select a professor", "featured-professor")}

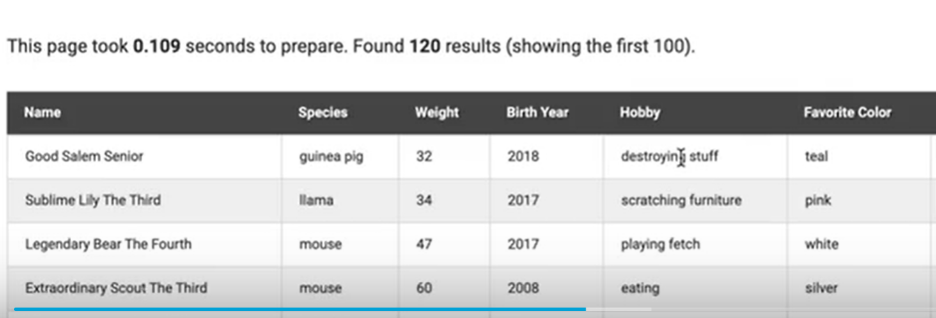
create a template in loco translate

* in setting

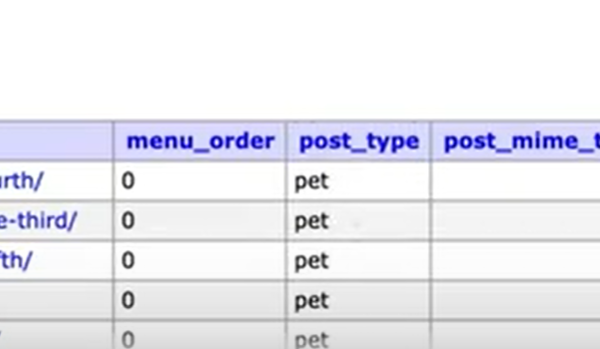


DB in plugin creation

* Post Types, use metadata for other properties of the post
* Use tables, custom SQL queries

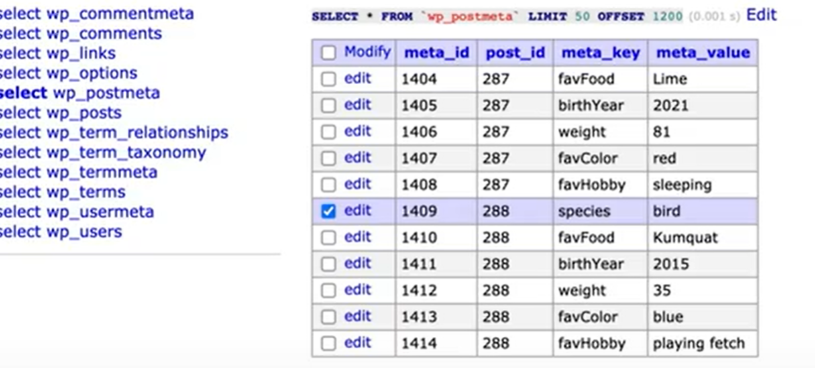


* Speed is similar if we get all of them
* But what about custom queries with conditions?
  + Db => 10 times faster
* Why custom table is faster?
  + **Post type** // even if we query metadata
    - **Strength** // It can store **any type** or shape of data by using the post-meta approach. Why for all posts is not that different? Because it just looks in the post table for the posts with that post\_type and then knows all the ids for that post\_type and in the metadata table filters those ids. It can index metadata based on id ( Indexing is the way to get an unordered table into an order that will maximize the query's efficiency while searching. ) and get a faster result.



* + - **Weakness** // for the large number of posts may be slow. Table **postmeta** is used even ACF and we need to search and create the relationship between the two tables.
    - Looking for only dogs, It can index the meta\_key, look for dogs, but it is not ideal when the metadata table becomes huge ( for each pet six rows in metadata, or products a lot of rows in metadata table)





* Custom table
  + Weakness // We lose a lot of cool free features. We should do everything ourselves, rest api, admin interface, permissions, …. Development time!
* Not always the right choice! 9 times out of 10, the custom post type is the smarter choice!
* **When to use a custom table?**
  1. Need to query by one custom metadata // not just store, search by it for
  2. many posts

global $wpdb

* $wpdb->insert(tablename, array)
* $wpdb->query($query)
* $wpdb->get\_results("")
* $wpdb->prepare() // prepare does not go to db, returns a string of text and we can feed that to get\_results for security
* $wpdb->delete($this->tablename, array('id' => $id));

Database Creation

* add\_action(**'activate\_new-database-table/new-database-table.php'**, array($this, 'onActivate'))

if (!defined('ABSPATH')) {

exit;

}

require\_once plugin\_dir\_path(\_\_FILE\_\_) . 'inc/generatePet.php';

class PetAdoptionTablePlugin

{

public $charset;

public $tablename;

public function \_\_construct()

{

global $wpdb;

$this->charset = $wpdb->get\_charset\_collate();

$this->tablename = $wpdb->prefix . "pets";

add\_action('activate\_new-database-table/new-database-table.php', array($this, 'onActivate'));

add\_action('admin\_head', array($this, 'onAdminRefresh'));

add\_action('wp\_enqueue\_scripts', array($this, 'loadAssets'));

add\_filter('template\_include', array($this, 'loadTemplate'), 99);

}

public function onActivate()

{

// to use dbDelta, it has a specific syntax

// echo $this->tablename;

// echo $this->charset;

require\_once(ABSPATH . 'wp-admin/includes/upgrade.php');

dbDelta("CREATE TABLE $this->tablename (

id bigint(20) unsigned NOT NULL AUTO\_INCREMENT,

birthyear smallint(5) NOT NULL DEFAULT 0,

petweight smallint(5) NOT NULL DEFAULT 0,

favfood varchar(60) NOT NULL DEFAULT '',

favhobby varchar(60) NOT NULL DEFAULT '',

favcolor varchar(60) NOT NULL DEFAULT '',

petname varchar(60) NOT NULL DEFAULT '',

species varchar(60) NOT NULL DEFAULT '',

PRIMARY KEY (id)

) $this->charset;"); }}

// Do not forget this!!! or it won't work!!

$PetAdoptionTablePlugin = new PetAdoptionTablePlugin();

Adding to db

* Admin\_head // action
* $wpdb->insert(tablename, array(prop=>value))

public function onAdminRefresh()

{

global $wpdb;

$wpdb->insert($this->tablename, array(

'birthyear' => 2015,

'petweight' => 10,

'petname' => 'Meowsalot',

'favfood' => 'apples',

'favcolor' => 'green',

'species' => 'cat',

'favhobby' => 'scratching furniture'

));

}

* add\_filter('template\_include', array($this, 'loadTemplate'), 99); // like the content filter but for template of a page

public function loadTemplate($template)

{

if (is\_page('pet-adoption')) {

return plugin\_dir\_path(\_\_FILE\_\_) . 'inc/template-pets.php';

}

return $template; // if not true return the default template for that slug

}

Security

* Sequel injection
* Sequel attacks
* If everything is hardcoded is okay.
* But if we send it through a json object, or in url.
* $wpdb->prepare()
* for var in the string, we need "" not ''
* for var in string we need " " not ' '

<?php

global $wpdb;

$tablename = $wpdb->prefix . 'pets';

// prepare does not go to db, returns a string of text and we can feed that to get\_results

//for var in string we need "" not ''

$ourQuery = $wpdb->prepare("SELECT \* FROM $tablename LIMIT 100", );

$pets = $wpdb->get\_results($ourQuery);

?>

<table class="pet-adoption-table">

<tr>

<th>Name</th>

<th>Species</th>

<th>Weight</th>

<th>Birth Year</th>

<th>Hobby</th>

<th>Favorite Color</th>

<th>Favorite Food</th>

</tr>

<?php

foreach ($pets as $pet) {?>

<tr>

<td><?php echo $pet->petname ?></td>

<td><?php echo $pet->species ?></td>

<td><?php echo $pet->petweight ?></td>

<td><?php echo $pet->birthyear ?></td>

<td><?php echo $pet->favhobby ?></td>

<td><?php echo $pet->favcolor ?></td>

<td><?php echo $pet->favfood ?></td>

</tr>

<?php

}

?>

</table>

global $wpdb;

// prepare does not go to db, returns a string of text and we can feed that to get\_results

$ourQuery = $wpdb->prepare("SELECT \* FROM wp\_pets WHERE species = %s AND birthyear > %d LIMIT 10", array('hamster', 2018));

$pets = $wpdb->get\_results($ourQuery);

var\_dump($pets)

Dynamic queries

* To keep the template file clean move queries to another file with a class to not worry about function names. Require it in the template.

foreach ($getPets->$pets as $pet) {?> // $pets a property of class

<?php

require\_once plugin\_dir\_path(\_\_FILE\_\_). 'GetPets.php';

$getPets = new Getpets();

get\_header();

?>

Dynamic values

class GetPets

{

public $pets;

public $args;

public $placeholders;

public $count;

public function \_\_construct()

{

global $wpdb;

$tablename = $wpdb->prefix . 'pets';

// $ourQuery = $wpdb->prepare("SELECT \* FROM $tablename WHERE species = %s LIMIT 100", array($\_GET['species']));

// $this->pets = $wpdb->get\_results($ourQuery);

$this->args = $this->getArgs(); // access it in other methods

$this->placeholders = $this->createPlaceholders();

//don't need to access it in other methods.

$query = "SELECT \* FROM $tablename ";

$countQuery = "SELECT COUNT(\*) FROM $tablename ";

$query .= $this->createWhereText();

$countQuery .= $this->createWhereText();

$query .= " LIMIT 100";

$this->count = $wpdb->get\_var($wpdb->prepare($countQuery,$this->placeholders));

$this->pets = $wpdb->get\_results($wpdb->prepare($query, $this->placeholders));

}

* There is a way to send one request and get the count too, but sending a query may be more performant.
* Create the argument for query and array of prepare

public function getArgs()

{

$temp = array(

'favcolor' => sanitize\_text\_field($\_GET['favcolor']),

'minyear' => sanitize\_text\_field($\_GET['minyear']),

'maxyear' => sanitize\_text\_field($\_GET['maxyear']),

'minweight' => sanitize\_text\_field($\_GET['minweight']),

'maxweight' => sanitize\_text\_field($\_GET['maxweight']),

'favhobby' => sanitize\_text\_field($GET['favhobby']),

'favfood' => sanitize\_text\_field($\_GET['favfood']),

'species' => sanitize\_text\_field($\_GET['species']),

'petname' => sanitize\_text\_field($\_GET['petname'])

);

// It is like javascript. As long as it has a value and it's not empty

return array\_filter($temp, function($x){

return $x; // return true and array\_filter returns the value});

public function createPlaceholders()

{

// in placeholder we don't need the property name, the name is needed for query

return array\_map(function($x){

return $x; // returns the value itself

},$this->args); }

public function createWhereText()

{

$whereQuery = "";

//if there is an arg include where

if(count($this->args)){

$whereQuery = "WHERE ";// don't forget the space! }

$currentPosition = 0; // index is string we need and except for the last item

foreach($this->args as $index => $item){ // we only want parameter name like species

// because we have maxweight or minweight,it becomes complicated we can use switch in another function

$whereQuery .= $this->specificQuery($index);

if($currentPosition != count($this->args) - 1){

$currentPosition++;

$whereQuery .= " AND "; }}

return $whereQuery; }

* Create the query string
* Switch for args minweight

function specificQuery($index){

//you can repeat the same property in the query

switch ($index){

case "minweight":

return "petweight >= %d";

case "maxweight":

return "petweight <= %d";

case "minyear":

return "birthyear >= %d";

case "maxyear":

return "birthyear <= %d";

default:

//for others, favhobby

return $index . " = %s";

}}

How to submit a form

* Include the logic inside the page, and by default, it is sent to its url
* Or use this way
  + When we post to the given url, wordpress looks for the input with name action and will create a hook createpet that we can hook on to in our main plugin file.

add\_action('admin\_post\_createpet', array($this, 'createPet'));

add\_action('admin\_post\_nopriv\_createpet', array($this, 'createPet'));

add\_action('activate\_new-database-table/new-database-table.php', array($this, 'onActivate'));

// add\_action('admin\_head', array($this, 'populateFast'));

add\_action('wp\_enqueue\_scripts', array($this, 'loadAssets'));

add\_filter('template\_include', array($this, 'loadTemplate'), 99);

}

public function createPet()

{

if (current\_user\_can('administrator')) {

echo 'lll';

$pet = generatePet();

$pet['petname'] = sanitize\_text\_field($\_POST['incomingpetname']);

global $wpdb;

$wpdb->insert($this->tablename, $pet);

wp\_redirect(site\_url('/pet-adoption'));

} else {

wp\_redirect(site\_url());

}}

* + This method is only for logedin users whether admin or not.

<?php

if (current\_user\_can('administrator')) { ?>

<form action="<?php echo esc\_url(admin\_url('admin-post.php')) ?>" class="create-pet-form" method="POST">

<p>Enter a name for a new pet. Its species, weight, and other details will be randomly generate.</p>

<input type="hidden" name="action" value="createpet">

<input type="text" name="incomingpetname" placeholder="name...">

<button>Add Pet</button>

</form>

<?php }

?>

Full site editing – Block theme

* Traditional approach => editing php template files, and pages.
* WordPress editor -> appearances -> full-site editing. No need for an editor. Good for end-user.
* Block Theme!

Block Theme

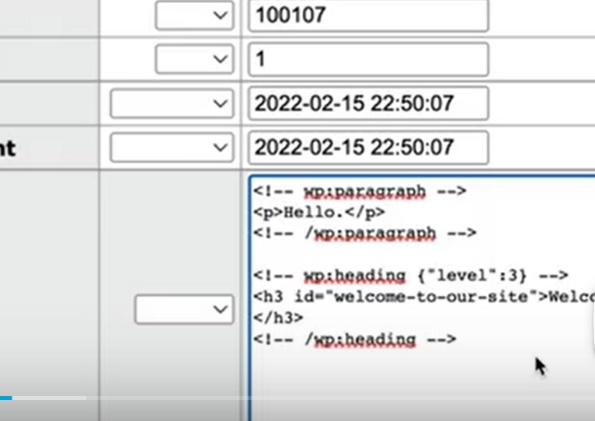
* Like a traditional theme a folder and file index.php // leave it blank
* A folder named templates
* Index.html //Everything in this file should be blocks, no top-level html. So the editor can treat it just like a regular WordPress post or page. Everything will be blocks in an editor like a normal page.

<!-- wp:paragraph -->

<p>Hello there</p>

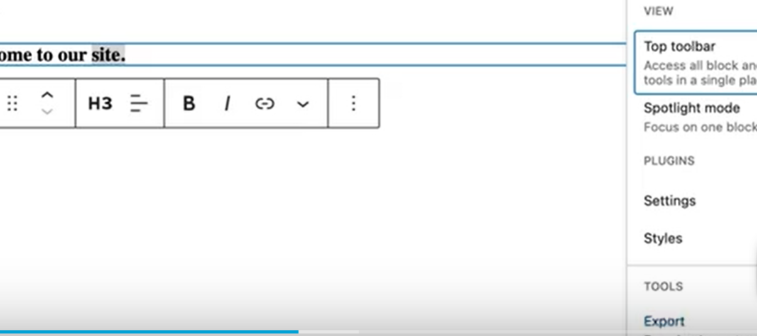
<!-- /wp:paragraph -->

* If we make changes in the editor, WordPress won't change this file, but store it as a post in db
* theme.json // if you don't include this file you may see a blank page

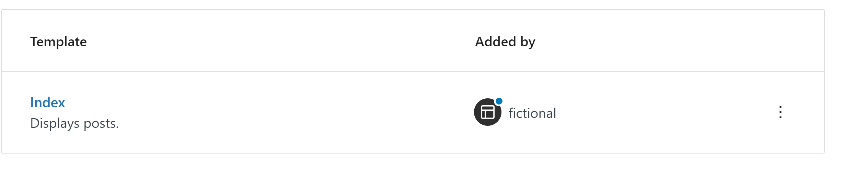


How to change the file

* Copy paste from db
* Export from admin

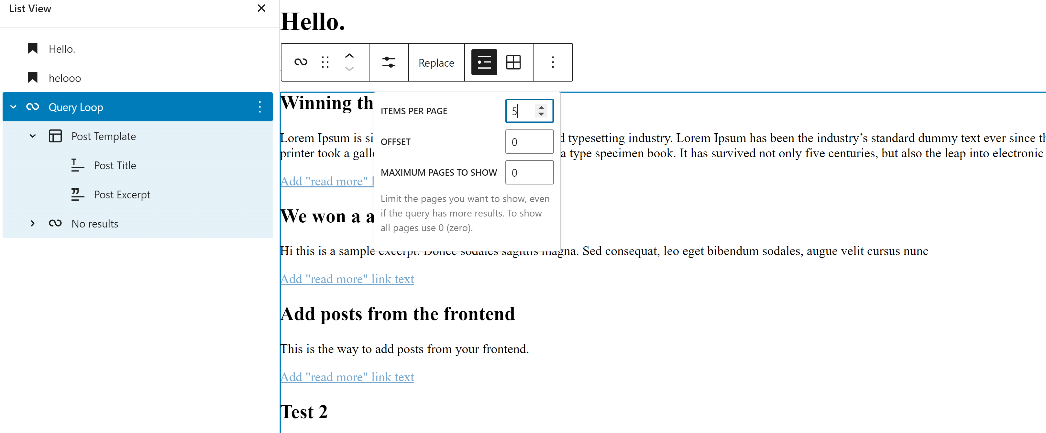


* Revert to default



How does it work?

* Index.html works like index.php in a normal theme
* Single? You need the same name in templates. With .html
* / // In the editor with this, we can add the
* Query block type // very basic
  + No meta values
  + Relationships
* Wp-includes -> blocks //Default wordpress blocks



* Block Patterns // Presets for blocks, tab trigger snippet in your text editor pattern. Arranging blocks. When using the pattern in a post editor for example, it doesn't store any link to the pattern, just blocks. That is problematic when you want to change the pattern and won't be affected where it is used.
* Stand-alone plugins like with did
* Theme-based blocks.
* wp.registerBlotType() in folder blocks, banner.js.
* You can use function.php, images, inc, src, template-parts, css folders from another theme.
* Remove the build
* @wordpress/scripts are needed, it will look for the template, index.html, but to make it look in the our-blocks folder and to have multiple output points. In JSON.

"start": "wp-scripts start sr/index.js our-blocks/banner.js",

* npm install // to install the packages
* npm run start
* in function.php

function bannerBlock()

{

wp\_register\_script('bannerBlockScript', get\_stylesheet\_directory\_uri() . '/build/banner.js', array('wp-blocks','wp-editor'));

register\_block\_type('ourblocktheme/banner', array(

'editor\_script' => 'bannerBlockScript'

));

}

add\_action('init', 'bannerBlock');

* in after\_setup\_theme callback // add css. Since it's a theme we load all css in the admin. This approach is good when you use tailwind css, you don't style each bock, but you use reusable utility classes. One global css file for both admin and front and WYSIWYG editor.

add\_theme\_support('editor-style');

//must be https, add the google font add\_editor\_style(array('https//fonts.googleapis.com/css?family=Roboto+Condensed:300,300i,400,400i,700,700i|Roboto:100,300,400,400i,700,700i','build/style-index.css', 'build/index.css'));

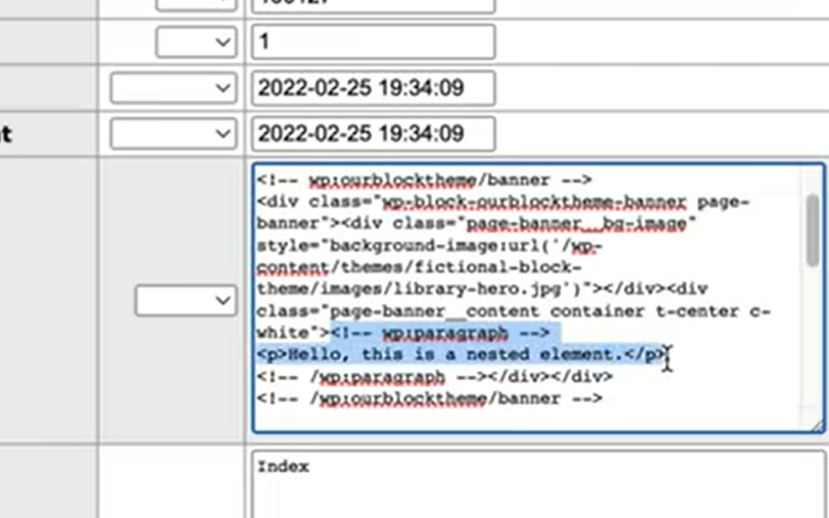
* Edit content
* We can use attributes for the title and button, but we cannot set dynamic order and layout.
* We need the style.css in the root for the front end.

Nested Blocks

* InnerBlocks // it is global in wp.blocEditor.InnerBlocks but import it instead to make it look modern. Wp official package has webpack configuration and sets different aliases when it sees the name and looks for global scope.

import {InnerBlocks} from "@wordpress/block-editor"

* <InnerBlocks allowedBlocks={[ "core/paragraph", "core/heading"]} /> // Edit callback
* <InnerBlocks.Content /> //save callback



Prevent repeating register block in php

* Create a function

class JSXBlock

{

public $name;

public function \_\_construct($name)

{

$this->name = $name;

add\_action('init', array($this,'onInit'));

}

public function onInit()

{

wp\_register\_script($this->name, get\_stylesheet\_directory\_uri() . "/build/{$this->name}.js", array('wp-blocks','wp-editor'));

register\_block\_type("ourblocktheme/{$this->name}", array(

'editor\_script' => $this->name

));

}

}

new JSXBlock('banner');

new JSXBlock('genericheading');

* Package json

"start": "wp-scripts start src/index.js our-blocks/banner.js our-blocks/genericheading.js",

* Custom inner blocks

<InnerBlocks allowedBlocks = {["ourblocktheme/genericheading"]} />

Keep track of the size and text of the inner heading

registerBlockType("ourblocktheme/genericheading", {

title: "Generic Heading",

attributes: {

**text: {type: "string"},**

**size: {type: "string", default: "large"}**

},

edit: EditComponent,

save: SaveComponent

} )

* RichText // activate an editable functionality for the text

function EditComponent(props){

function handleTextChange(x){

//s is needed attributes

props.setAttributes({text: x})

}

return(

<>

<RichText value={props.attributes.text} onChange={handleTextChange} />

</>

)

}

import {RichText} from "@wordpress/block-editor"

* tagName="h1" // to let the richtext know what tag we want
* className // to show it visually
* allowedFormats = {["core/bold" , "core/italic"]} // allow options in the toolbar for the block
* BlockControls add the small, large, med

import {toolbarGroup, ToolBarButton} from "@wordpress/components"

* Size Buttons
* BlockControls -> ToolbarGroup -> ToolbarButton
  + isPressed // two show it visually

<BlockControls>

<ToolbarGroup>

<ToolbarButton isPressed={props.attributes.size === "large"} onClick={()=> props.setAttributes({size: "large"})}>Large</ToolbarButton>

<ToolbarButton isPressed={props.attributes.size === "medium"} onClick={()=> props.setAttributes({size: "medium"})}>Medium</ToolbarButton>

<ToolbarButton isPressed={props.attributes.size === "small"} onClick={()=> props.setAttributes({size: "small"})}>Small</ToolbarButton>

</ToolbarGroup>

</BlockControls>

import {RichText, BlockControls} from "@wordpress/block-editor"

import {ToolbarGroup, ToolbarButton} from "@wordpress/components"

* Save function

function createTagName(){

switch(props.attributes.size){

case "large":

return "h1"

case "medium":

return "h2"

case "small":

return "h3"

}}

return (

<RichText.Content value={props.attributes.text} tagName={createTagName()} className={`headline headline--${props.attributes.size}`} />

)

Theme.json

* Check the documentation for the property names
* Customizing wrapper div **size**
  + Use version
    - Setting -> Layout -> contentSize (840 normal) // This will make
    - this is a **global setting**

{

"version": 2,

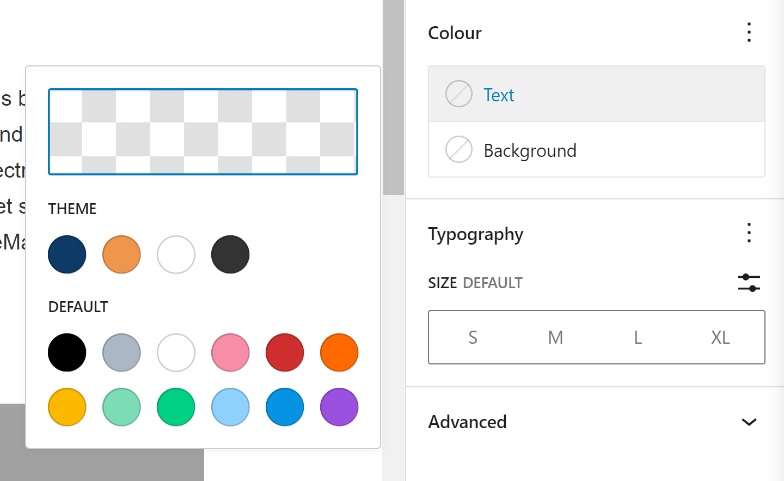
"settings": {

"layout": {

"contentSize": "800px"

}}}

* **Supports**: **Block widths** // In js and registerBlockType
* The color palate for the theme
  + Slug // Computer-friendly name for the palette



supports: {

align: ["full"]

},

attributes: {

align: {type: "string", default: "full"}

},

"settings": {

"color": {

"palette": [

{

"slug": "primary",

"color": "#0d3b66",

"name": "Primary"

},

{

"slug": "secondary",

"color": "#ee964b",

"name": "Secondary"

},

{

"slug": "background",

"color": "#FFFFFF",

"name": "Secondary"

},

{

"slug": "foreground",

"color": "#333333",

"name": "Foreground"

}]}, …

* Styles for blocks // default styles

"styles": {

"blocks": {

"core/button": {

"color": {

"text": "#FFFFFF",

"background": "var(--wp--preset--color--primary)"

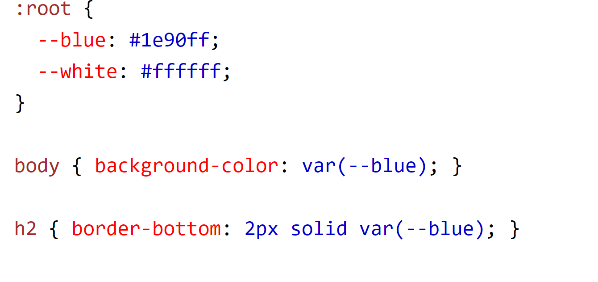
}

}

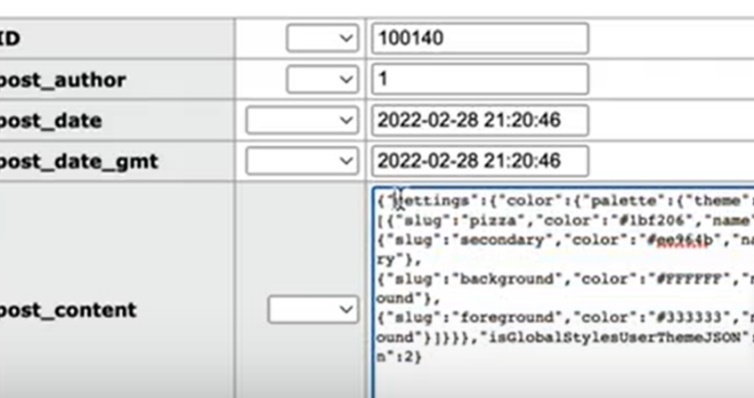
}

},

* + Css variables



* In the editor the black-white icon, and click colors // it has pallet options and the non-programmer can change them. In the db posts, the theme json is stored in post\_content of wp\_global\_styles post type, and it takes it from there, this option will not change theme.json and can be reverted to default in the editor.



* Per-block settings

"settings": {

"blocks": {

"core/button": {

"border": {

"color": false,

"radius": false,

"style": false,

"width": false

}

}

},

* General setting // remove options from the block right sidebar menu

"settings": {

"typography":{

"fontSize": []

},

* Using wp as a Cms probably has no need for theme.json, but as for page builder or design-builder

Custom block button

* In the saveComponent method, we don't need the RichText anymore, and no inner tag.
* To search for URLs to set the link
  + npm install @wordpress/icons // Icon is not installed by default

import {link } from "@wordpress/icons"

* **Popover**

import {ToolbarGroup, ToolbarButton, Popover, **Button**} from "@wordpress/components"

* **ToolbarButton**

// we don't want to store it in db, use state

const [isLinkPickerVisible, setIsLinkPickerVisible] = useState(false)

function buttonHandler(){

// the opposite of previous value to toggle

setIsLinkPickerVisible(prev => !prev)

}

<ToolbarGroup>

<ToolbarButton onClick={buttonHandler} icon={link} />

</ToolbarGroup>

* **LinkControl**

import {RichText, BlockControls, \_\_experimentalLinkControl as LinkControl} from "@wordpress/block-editor"

* **Show the popover**

{isLinkPickerVisible && (

<Popover position="middle center">

<LinkControl settings={[]} value={props.attributes.linkObject} onChange={handleLinkChange} />

<Button variant="primary" onClick={()=> setIsLinkPickerVisible(false)} style={{display: "block", width: "100%"}}>Confirm Link</Button>

</Popover>

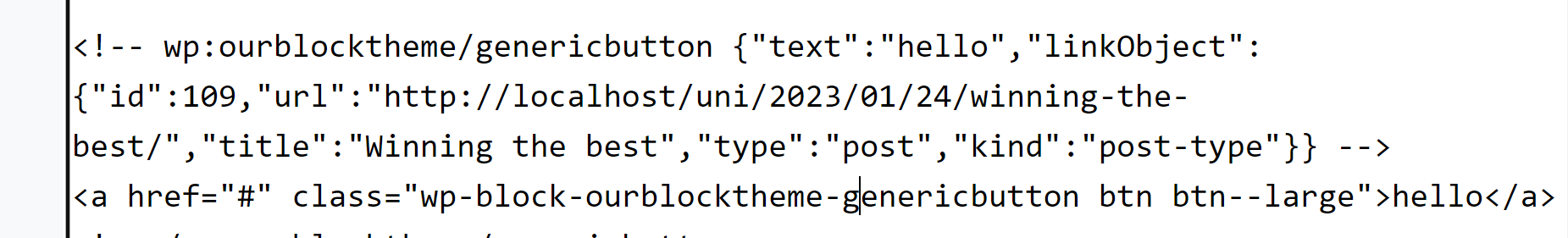
)

* Update the new link

function handleLinkChange(newLink){

props.setAttributes({linkObject: newLink})

}

* Save the link

return(

<a href={props.attributes.linkObject.url} className={`btn btn--${props.attributes.size}`}>

{props.attributes.text}

</a>

)

* ColorPicker

import {RichText, **InspectorControls**, BlockControls, \_\_experimentalLinkControl as LinkControl} from "@wordpress/block-editor"

import {ToolbarGroup, ToolbarButton, Popover, Button, **PanelBody, PanelRow, ColorPalette**} from "@wordpress/components"

* Attribute

colorName: {type: "string", default: "blue"}

* To store only the name
* Disable some options
  + disableCustomColors

<InspectorControls>

<PanelBody title="Color" initialOpen={true}>

<PanelRow>

<ColorPalette disableCustomColors = {true} clearable = {false} colors={ourColors} value={currentColorValue} onChange={handleColorChange} />

</PanelRow>

</PanelBody>

</InspectorControls>

* + clearable

import {RichText, InspectorControls, BlockControls, \_\_experimentalLinkControl as LinkControl, **getColorObjectByColorValue**} from "@wordpress/block-editor"

* Event functions // store colors name to create an abstraction layer

function handleColorChange(colorCode){

// from the hex value that the color palette gives us, we need to find its color name

const {name} = getColorObjectByColorValue(ourColors, colorCode)

console.log(name)

props.setAttributes({colorName: name})

}

const currentColorValue = ourColors.filter(color => {

return color.name == props.attributes.colorName

})[0].color

* Create a new file for your colors in js in inc folder

const ourColors = [

{name: "blue", color: "#0d3b66"},

{name: "orange", color: "#ee964b"},

{name: "dark-orange", color: "#f95738"},

]

export default ourColors

import ourColors from '../inc/ourColors'

* program link on banner // since the program is just a slug post type not a page, to add it write /programs, or the whole https://...

Php render callback

* If we change the html for SaveComponent we need to keep track of historic HTML, so the wp won't ask for recovery but still, we need to click on the update button for all the posts. The html output in SaveComponent() is stored in db and is not dynamic. Php render callback helps with that.
* **$content** // for nested blocks, to access them
* Create a file for rendering callback html
* We need to return the nested blocks in SaveComponents because we need to store the content given by the user in db

function SaveComponent(){

return (<InnerBlocks.Content />)}

<div class="page-banner">

<div class="page-banner\_\_bg-image" style="background-image: url('<?php echo get\_theme\_file\_uri('/images/library-hero.jpg') ?>')"></div>

<div class="page-banner\_\_content container t-center c-white">

<?php echo $content ?>

</div>

</div>

* Php file

class JSXBlock

{

public $name;

public $renderCallback;

public function \_\_construct($name, $renderCallback = null)

{

$this->name = $name;

$this->renderCallback = $renderCallback;

add\_action('init', array($this,'onInit')); }

function ourRenderCallback($attributes, $content){

// the var {} works only with ""

// To have unique rendercallbacks for each block

ob\_start(); // anything in between will be treated as a big string of text

require get\_theme\_file\_path("/our-blocks/{$this->name}.php");

return ob\_get\_clean();}

public function onInit()

{

wp\_register\_script($this->name, get\_stylesheet\_directory\_uri() . "/build/{$this->name}.js", array('wp-blocks','wp-editor'));

$ourArgs = array(

'editor\_script' => $this->name

);

$this->renderCallback == true ? $ourArgs['render\_callback'] = [$this, 'ourRenderCallback'] : '';

register\_block\_type("ourblocktheme/{$this->name}",$ourArgs);

}}

new JSXBlock('banner', true); // make php render callback optional

new JSXBlock('genericheading');

new JSXBlock('genericbutton');

* no html is stored in db



Background image choosing functionality

* InspectorControls
* MediaUpload // it gives us an object so we have de-structure it.
  + Open//It will be used to open the media upload window
* MediaUploadCeck // current user had permission to upload media

**import { Button, PanelBody, PanelRow } from "@wordpress/components"**

import {InnerBlocks, **InspectorControls**, **MediaUpload**, **MediaUPloadCheck**} from "@wordpress/block-editor"

* apiFetch // To access the media base on id, in PHP we coded the images to be stored in custom sizes now we need the size with pageBanner name
  + http://localhost/uni/wp-json/wp/v2/media

import apiFetch from "@wordpress/api-fetch"

* useEffect // To show the background when chosen,

import {useEffect} from "@wordpress/element"

* attributes

attributes: {

align: {type: "string", default: "full"},

imgID: {type: "number"},

imgURL: {type: "string"}

},

* functions

useEffect(function(){

async function go(){

// console.log('this is done')

const response = await apiFetch({

path: `/wp/v2/media/${props.attributes.imgID}`,

method: "GET"

})

// not to look up again in php render, store it in an attribute

console.log(response.media\_details.sizes.pageBanner.source\_url)

// console.log(`/wp/v2/media/${props.attributes.imgID}`)

props.setAttributes({imgURL: response.media\_details.sizes.pageBanner.source\_url})

}

// You have to call it!!!

go()

},[props.attributes.imgID])

function onFileSelect(x){

// console.log(x.id)

props.setAttributes({imgID: x.id})

}

* JSX //add a right panel setting for MediaUploadCheck
* onSelect={} // called whenever the user selects a file
* Use the id of the object returned which is the id of the piece of media we uploaded.

<InspectorControls>

<PanelBody title="Background" initailOpen={true}>

<PanelRow>

<MediaUploadCheck>

<MediaUpload onSelect={onFileSelect} value={props.attributes.imgID} render={({open})=>{

return <Button onClick={open}>Choose Image</Button>

}} />

</MediaUploadCheck>

</PanelRow>

</PanelBody>

</InspectorControls

* Show in edit // use the props

<div className="page-banner\_\_bg-image" style={{backgroundImage: `url('${props.attributes.imgURL}')`}}></div>

* In php // set a default

if(!$attributes['imgURL']){

$attributes['imgURL'] = get\_theme\_file\_uri('/images/library-hero.jpg');

}

* wp\_localize\_scripts //This will add a js var. get dynamic theme-based url in js, to add a default image to js

class JSXBlock

{

…

public $data;

public function \_\_construct($name, $renderCallback = null, $data = null)

{

$this->data = $data; …

}

public function onInit()

{

wp\_register\_script($this->name, get\_stylesheet\_directory\_uri() . "/build/{$this->name}.js", array('wp-blocks','wp-editor'));

if($this->data){

// to localize some json with the script we registered above

// the second arg is the var name we can access in js

new JSXBlock('banner', true, ['fallbackimage' => get\_theme\_file\_uri('/images/li')]);

wp\_localize\_script($this->name, $this->name, $this->data);

}

…

}}

new JSXBlock('banner', true, ['fallbackimage' => get\_theme\_file\_uri('/images/library-hero.jpg')]);

imgURL: {type: "string", default: window.banner.fallbackimage}

Static placeholder blocks (Events and Blogs section)

* Columns // default 2020 theme block in WordPress. We can add custom classes to them, but still, we are tied to WordPress media query classes. No activity on the editor. But to include in other pages.
* No JSX. Basic react code. No need to update package.json
* wp.blocks.registerBlockType // because we don't use package.json workflow to be processed
* wp.element.createElement() // create a component without JSX
* wp.element.creatElement(type of element,props, content)
* the js

wp.blocks.registerBlockType("ourblocktheme/eventsandblogs", {

title: "Events and Blogs",

edit: function(){

return wp.element.createElement("div", {className: "our-placeholder-block"}, "Events and Blogs Placeholder")

},

save: function(){

return null }})

* function.php

class PlaceholderBlock

{

public $name;

public function \_\_construct($name)

{

$this->name = $name;

add\_action('init', array($this,'onInit'));}

function ourRenderCallback($attributes, $content){

ob\_start(); // anything in between will be treated as a big string of text

require get\_theme\_file\_path("/our-blocks/{$this->name}.php");

return ob\_get\_clean();}

public function onInit()

{ // no need for build version, no jsx

wp\_register\_script($this->name, get\_stylesheet\_directory\_uri() . "/our-blocks/{$this->name}.js", array('wp-blocks','wp-editor'));

register\_block\_type("ourblocktheme/{$this->name}",array(

'editor\_script' => $this->name,

'render\_callback' => [$this, 'ourRenderCallback'])); }}

new PlaceholderBlock("eventsandblogs");

* php file // the same as the normal theme

Standard way for Global Header and Footer

* The official way like the twenty-twenty theme is to use parts and include the theme in templates where we want. If you want to use your own css and html and just one block then you don't need the standard. You can use the placeholder method mentioned.

Multiple of the same block (slideshow)

* Slideshow content is the same as a banner for 80%, with a few differences in slideshow.php, slide.php (taken from banner.php), and edit and save of slideshow.js for nested slides. Fix the heroslider javascript. Point towards building index.js in function.php.

Resilient default index.html

* Index.html // make it resilient. Since the url may be different, delete the banner image url from index.html copied from db. For the slideshow:

new JSXBlock('slide', true, **['themeimagepath' => get\_theme\_file\_uri('/images/')])**;

* In index.html

<!-- wp:ourblocktheme/slide {"themeimage":"apples.jpg"} -->

* In slide.js

useEffect(function(){

if( props.attributes.themeimage){

console.log(`${slide.themeimagepath}${props.attributes.themeimage}`)

// it should be saved to get affected

props.setAttributes({imgURL: `${slide.themeimagepath}${props.attributes.themeimage}`})

}},[]) // [] means only initial load

attributes: {

themeimage: {type: "string"},

* When uploading imageurl, set themeimage back to null.

props.setAttributes({themeimage: "", imgURL: response.media\_details.sizes.pageBanner.source\_url})

* In slide.php

if($attributes['themeimage']){

$attributes['imgURL'] = get\_theme\_file\_uri('/images/' . $attributes['themeimage'] );

}

* Single post placeholder // simple like others
* There is a hierarchy for chosen files, for any slug that does not have a template

Restrict different block types

* In blog editor vs Full-site editor

function myallowedblocks($allowed\_block\_types, $editor\_context){

if($editor\_context->post->post\_type == "professor"){

return array('core/paragraph', 'core/list')

}

// deafutl block types allowed => allowed\_block\_types\_all

// page/post editor screen

if(!empty($editor\_context->post)){

return $allowed\_block\_types;

}

//FSE Screen

return array('ourblocktheme/header', 'ourblocktheme/footer')

}

add\_filter('allowed\_block\_types\_all' , 'myallowedblocks', 10, 2);

Blank Template

Security

* Updated //Keep the WordPress updated
  + Automatic updates // this is always better
  + Disable auto-update // test local, someone needs to test everything in a short period.
* By default WordPress only enables minor updated
* Major updates Automatically
* How to make that happen with git
  + Only keep the theme folder in your repo
  + Or deploy certain folders by setting the deploy service
  + .gitingnore

Custom Query Vars

* Wordpress by default ignores any custom properties in the url. It looks for s(search) and ….
* We need to register them to be able to use them.
* In function.php

function universityQueryVars($vars){

//register a new query var

$vars[] = 'skyColor';

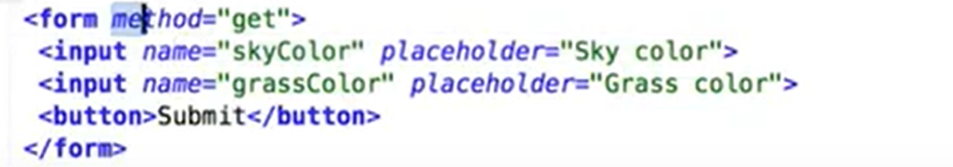
$vars[] = 'grassColor';

return $vars;

}

add\_filter('query\_vars', 'universityQueryVars')

* Sanitize\_text\_field**(get\_query\_var('skyColor'))** == 'blue' // To use
* We can use this to have forms that have actions on the same page
  + Get // to share the URL with parameters



* It can be used in custom queries