

Day 2

Customize Theme and Admin

**BY:**

**BRUNO ANGELO MEDEIROS**bruno.medeiros@sc.senai.br

# ADMIN

## CUSTOMIZING LOGIN

For this, we’ll create a plugin, the advantege of using a plugin is that we can drop it into any WordPress site we develop.

### SETTING UP THE PLUGIN

Create a file called **customize-theme-and-admin.php**, and add the initial code**:**

1. <?php
2. /\*\*
3. \* Plugin Name: Customize Theme and Admin
4. \* Plugin URI: http://yourdomain.com
5. \* Description: It customizes the WordPress.
6. \* Version: 1.0
7. \* Author: Your Name
8. \* Author URI: http://yourdomain.com
9. \* License: GPLv2
10. \*/

### CUSTOM LOGO

To add our own logo to the login page, we need few steps:

1. First, create a folder called **media** inside our plugin folder, and upload our logo to it.
2. In the plugin file, add the following function, attaching it to the **login\_enqueue\_scripts** action hook:
   1. // add a new logo to the login page
   2. **function** my\_login\_logo() { ?>
   3. <style type="text/css">
   4. .login #login h1 a {
   5. background-image: url( <?= plugin\_dir\_url( **\_\_FILE\_\_** ) . 'media/logo.png'; ?> );
   6. }
   7. </style>
   8. <?php }
   9. add\_action( 'login\_enqueue\_scripts', 'my\_login\_logo' );

Of course, we can also completely customize the login page via CSS. If we need to do more changes, it might be best to enqueue custom styles sheets, like below:

1. **function** my\_login\_stylesheet() {
2. wp\_enqueue\_style( 'custom-login', plugin\_dir\_url( \_\_FILE\_\_ ) . 'style-login.css' );
3. wp\_enqueue\_script( 'custom-login', plugin\_dir\_url( \_\_FILE\_\_ ) . 'style-login.js' );
4. }
5. add\_action( 'login\_enqueue\_scripts', 'my\_login\_stylesheet' );

## DASHBOARD

### REMOVE METABOXES

We can remove any metaboxes that we don’t want. This will only apply to uers with a role lower than “administrator”.

For example, to remove those metaboxes for users other than administrators, add the following to our plugin:

1. // remove unwanted dashboard widgets for relevant users
2. **function** remove\_dashboard\_widgets() {
3. $user = wp\_get\_current\_user();
4. **if** ( ! $user->has\_cap( 'manage\_options' ) ) {
5. remove\_meta\_box( 'dashboard\_right\_now', 'dashboard', 'normal' );
6. remove\_meta\_box( 'dashboard\_activity', 'dashboard', 'normal' );
7. remove\_meta\_box( 'dashboard\_quick\_press', 'dashboard', 'side' );
8. remove\_meta\_box( 'dashboard\_primary', 'dashboard', 'side' );
9. }
10. }
11. add\_action( 'wp\_dashboard\_setup', 'remove\_dashboard\_widgets' );

### ADD CUSTOM WIDGETS

To add widgets metaboxes to the dashboard we need to do two steps:

1. Use the **wp\_add\_dashboard\_widget()** function to define the widge’ts parameters – its ID, the title, and the callback function. Activate this via the **wp\_dashboard\_setup** hook.
2. Write the call-back function to define the contente of the metabox.

In our plugin add for example:

1. // add new dashboard widgets
2. **function** add\_dashboard\_widgets() {
3. wp\_add\_dashboard\_widget( 'dashboard\_welcome', 'Welcome', 'add\_welcome\_widget' );
4. wp\_add\_dashboard\_widget( 'dashboard\_links', 'Useful Links', 'add\_links\_widget' );
5. }
6. **function** add\_welcome\_widget(){ ?>
8. This content management system lets you edit the pages **and** posts on your website.
10. Your site consists of the following content, which you can access via the menu on the left:
12. <ul>
13. <li><strong>Pages</strong> - **static** pages which you can edit.</li>
14. <li><strong>Posts</strong> - news **or** blog articles - you can edit these **and** add more.</li>
15. <li><strong>Media</strong> - images **and** documents which you can upload via the Media menu on the left **or** within each post **or** page.</li>
16. </ul>
18. On each editing screen there are instructions to help you add **and** edit content.
20. <?php }
22. **function** add\_links\_widget() { ?>
24. Some links to resources which will help you manage your site:
26. <ul>
27. <li><a href="http://wordpress.org">The WordPress Codex</a></li>
28. <li><a href="http://easywpguide.com">Easy WP Guide</a></li>
29. <li><a href="http://www.wpbeginner.com">WP Beginner</a></li>
30. </ul>
31. <?php }
32. add\_action( 'wp\_dashboard\_setup', 'add\_dashboard\_widgets' );

### ADMINISTRATION FOOTER

We can change the message in the WordPress footer, where it usually says “thank you for creating with WordPress”.

Open the **Appearance** menu and click on **Editor**. Click on **functions.php** on the right side of our screen. We can also access the footer by using an FTP client to locate */wp-content/themes/NAME\_OF\_OUR\_THEME/functions.php*.

Or instead of editing the functions.php, we can add in our previous plugin file.

Now, add the following lines of code:

1. **function** change\_admin\_footer(){
2. echo '<span id="footer-note">From your friends at <a href="http://www.yourdomain.com/" target="\_blank">CMS Training</a>.</span>';
3. }
4. add\_filter('admin\_footer\_text', 'change\_admin\_footer');

### REMOVE COLUMNS IN LISTING SCREEN

In case we need to remove some columns from listings screen, we use one of three action hooks:

* **manage\_posts\_columns**, for the posts listing screen
* **manage\_pages\_columns**, for the pages listing screen
* **manage\_$post\_type\_posts\_columns**, for a screen listing our custom post type. So for example if we had a 'cars' custom post type, we would use the **manage\_cars\_post\_type\_columns** action hook

Add the following in our **functions.php** or in a plugin file:

1. // remove tags from posts listing screen
2. **function** remove\_posts\_listing\_tags( $columns ) {
3. unset( $columns[ 'tags' ] );
4. **return** $columns;
5. }
6. add\_filter( 'manage\_posts\_columns', 'remove\_posts\_listing\_tags' );

## MENU

### RENAME

Not always the menu from Wordpress matches with the theme, so if we need to rename, add the following code to our plugin:

1. // Rename Posts to News in Menu
2. **function** change\_post\_menu\_label() {
3. **global** $menu;
4. **global** $submenu;
5. $menu[5][0] = 'News';
6. $submenu['edit.php'][5][0] = 'News Items';
7. $submenu['edit.php'][10][0] = 'Add News Item';
8. }
9. add\_action( 'admin\_menu', 'change\_post\_menu\_label' );

This code references the **$menu** and **$submenu** global variables - the numbers in square brackets tell WordPress which menu items we’re referring to, as each menu item has its own number which corresponds with its order in the menu.

The next step is to rename the submenu items for posts, and any other use of the term 'post' in the admin screens.

In our plugin, add the following:

1. // Edit submenus
2. **function** change\_post\_object\_label() {
3. **global** $wp\_post\_types;
4. $labels = $wp\_post\_types['post']->labels;
5. $labels->name = 'News';
6. $labels->singular\_name = 'News Item';
7. $labels->add\_new = 'Add News Item';
8. $labels->add\_new\_item = 'Add News Item';
9. $labels->edit\_item = 'Edit News Item';
10. $labels->new\_item = 'News Item';
11. $labels->view\_item = 'View News Item';
12. $labels->search\_items = 'Search News Items';
13. $labels->not\_found = 'No News Items found';
14. $labels->not\_found\_in\_trash = 'No News Items found in Trash';
15. }
16. add\_action( 'admin\_menu', 'change\_post\_object\_label' );

### REMOVE

To remove some menus that a certain role can’t use, we add the following:

1. // Remove Comments menu item for all but Administrators
2. **function** remove\_comments\_menu\_item() {
3. $user = wp\_get\_current\_user();
4. **if** ( ! $user->has\_cap( 'manage\_options' ) ) {
5. remove\_menu\_page( 'edit-comments.php' );
6. }
7. }
8. add\_action( 'admin\_menu', 'remove\_comments\_menu\_item' );

### REORDER

Some menu items its more frequenly used. So, to do this we use two filters:

* **custom\_menu\_order**, which activates the menu\_order filter and must have \_\_return\_true as its value.
* **menu\_order**, to which we attach our function defining the new menu order.

In we plugin, add the following:

1. // Move Pages above Media
2. **function** change\_menu\_order( $menu\_order ) {
3. **return** **array**(
4. 'index.php',
5. 'edit.php',
6. 'edit.php?post\_type=page',
7. 'upload.php',
8. );
9. }
10. add\_filter( 'custom\_menu\_order', '\_\_return\_true' );
11. add\_filter( 'menu\_order', 'change\_menu\_order' );

## COLOR SCHEME

To create a new scheme, all we need is a new CSS stylesheet. In this example, we’ll call it admin.css and place it in a folder customize-theme-and-admin/css. Edit the **customize-theme-and-admin.php** file and add this code:

1. // Custom WordPress Admin Color Scheme
2. **function** admin\_css() {
3. wp\_enqueue\_style( 'admin\_css', plugin\_dir\_url( **\_\_FILE\_\_** ) . 'css/admin.css' );
4. }
5. add\_action('admin\_print\_styles', 'admin\_css' );

# THEME

## CHILD-THEME

With child-theme we can ad dor edit styles, functions, page templates and more without the need to modify the core files of our theme.

When creating a child-theme we inherit all the functions of another theme and can modify them through a completely new file.

### GETTING STARTED

First, we need to create a directory inside the folder **wp-content/themes**. The WordPress recommends are that we use the name of the parent theme using the suffix “-child” just for the sake of organization.

Next we must create inside of the child theme’s directory a **style.css** file.

1. /\*\*
2. \* Theme Name:   Twenty Seventeen Child
3. \* Theme URI:    http://example.com/twenty-seventeen-child/
4. \* Description:  Twenty Seventeen Child Theme
5. \* Author:       John Doe
6. \* Author URI:   http://example.com
7. \* Template:     twentyseventeen
8. \* Version:      1.0.0
9. \* License:      GNU General Public License v2 or later
10. \* License URI:  http://www.gnu.org/licenses/gpl-2.0.html
11. \* Tags:         light, dark, two-columns, right-sidebar, responsive-layout, accessibility-ready
12. \* Text Domain:  twenty-seventeen-child
13. \*/

Here is what each line means:

* **Theme name**

This is the name that will show up for our theme in the WordPress back end.

* **Theme URI**

This points to the website or demonstration page of the theme at hand. This or the author’s URI must be present in order for the theme to be accepted into the WordPress directory.

* **Description**

This description of our theme will show up in the theme menu when we click on “Theme Details.”

* **Author**

This is the author’s name.

* **Author URI**

We can put our website’s address here if we want.

* **Template**

This part is crucial. Here goes the name of the parent theme, meaning its folder name. Be aware that it is case-sensitive, and if we don’t put in the right information, we will receive an error message, so double-check!

* **Version**

This displays the version of our child theme. Usually, we would start with 1.0.

* **License**

This is the license of our child theme. WordPress themes in the directory are usually released under a GPL license; we should stick with the same license as our parent theme.

* **License URI**

This is the address where our theme’s license is explained. Again, stick with what our parent theme says.

* **Tags**

The tags help others find our theme in the WordPress directory. Thus, if we include some, make sure they are relevant.

* **Text domain**

This part is used for internationalization and to make themes translatable. This should fit the “slug” of our theme.

But, for us only we need is the theme name and template, so for this reason our child theme’s header can be like below.

1. /\*\*
2. \* Theme Name:   Twenty Seventeen Child Theme
3. \* Description:  A child theme of the Twenty Seventeen default WordPress theme
4. \* Author:       Your Name
5. \* Template:     twentyseventeen
6. \* Version:      1.0.0
7. \*/

### ACTIVATE

Once our folder and style sheet are present, go to **Appearance**, then **Themes** in the WordPress find our child theme there. When we click on **Theme Details** now, we will see the contents of the style sheet header. That’s what that info is for.

At this point our child theme works just fine. If we activate it and reload the page all our content will be there but, it will have no styling information.

### MODIFY THEME’S CSS

#### FUNCTION.PHP

In our case we do have a stylesheet, so WordPress figures it shouldn’t load the parent file’s. To make sure we load the parent file’s stylesheet we will need to enqueue it. This can be done in the theme’s **functions.php** file. In this file, copy-paste the following code:

1. add\_action( 'wp\_enqueue\_scripts', 'enqueue\_parent\_styles' );
2. **function** enqueue\_parent\_styles() {
3. wp\_enqueue\_style( 'parent-style', get\_template\_directory\_uri().'/style.css' );
4. }

### OTHER FILES

We can not only target individual style declarations via the style sheet, but also override entire components of the parent theme.

For example, for every theme file present in the parent directory, WordPress will check whether a corresponding file is present in the child theme and, if so we want to make changes to the header, copy-paste the parent theme’s **header.php** file into our child theme.

#### ADD A CUSTOM SIDEBAR

In our example we’re using the Twenty Seventeen Theme, but by default, Twenty Seventeen doesn’t add sidebars to the front page section or to any pages for. It does display a sidebar on individual posts only.

First, we’ll need to copy some file to our child-theme, find these files by going to **\wp-content\themes\twentyseventeen\**. In that directory, we’ll see **front-page.php** and **page.php** files.

##### **ADD SIDEBAR CODE TO EACH TEMPLATE FILE.**

First, open up **functions.php** and add the code below:

1. // sidebar pages
2. **function** twentyseventeen\_body\_classes\_child( $classes ){
3. **if** ( is\_active\_sidebar( 'sidebar-1' ) &&  is\_page() ) {
4. $classes[] = 'has-sidebar';
5. }
6. **return** $classes;
7. }
8. add\_filter( 'body\_class', 'twentyseventeen\_body\_classes\_child' );

Second, open up **page.php**, and drop the following code replace before the closing **main** tag:

1. **</main>**<!-- #main -->
2. **</div>**<!-- #primary -->
3. **<?php** get\_sidebar(); **?>**
4. **</div>**<!-- .wrap -->
6. **<?php** get\_footer();

Now, open up **front-page.php**, and wrap the tag **div#primary** with a **div** element using the class **.wrap,** after the closing **div** tag add **<?php get\_sidebar(); ?>.** The code should be like this:

1. <!-- // header from the front-page.php -->
2. get\_header(); **?>**
3. **<div** class="wrap"**>**
4. **<div** id="primary" class="content-area"**>**
5. **<main** id="main" class="site-main" role="main"**>**
7. <!-- // All content from the front-page.php -->
9. **</main>**<!-- #main -->
10. **</div>**<!-- #primary -->
11. **<?php** get\_sidebar(); **?>**
12. **</div>**
13. **<?php** get\_footer();

Finally, go back to our **style.css** and add the following code to fix our changes:

1. .twentyseventeen-front-page .site-content{
2. **padding-top**: 5.5em;
3. }
5. .twentyseventeen-front-page .panel-content .wrap{
6. **padding-top**:0;
7. **padding-left**: 1em;
8. **padding-right**: 1em;
9. }