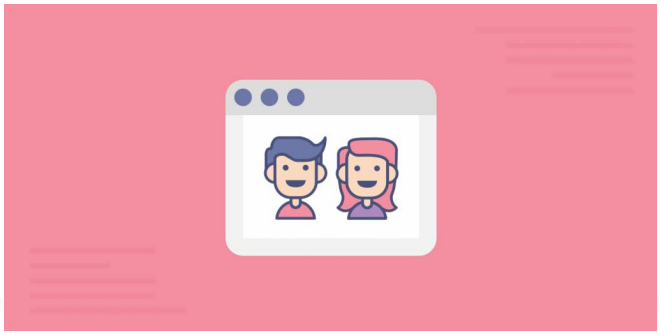


Siempre que un visor de sitio se inscriba para una cuenta, o usted agregue manualmente un nuevo usuario en la pantalla usuarios del dashboard, WordPress registra datos de usuario esenciales en tablas wp_users y wp_usermeta. El nuevo usuario recibe el nombre de usuario de su elección, una contraseña, una dirección de correo electrónico, una función que determina lo que puede ver en el panel de admin. Pero, ¿qué son exactamente los roles de usuario de WordPress?

Roles de Usuario y Capacidades de WordPress

El sistema de gestión de usuarios de WordPress se basa en dos conceptos clave: Funciones y capacidades.

Un Rol identifica un grupo de usuarios que tienen permiso para ejecutar las mismas tareas en el sitio web. Una Capacidad es la capacidad (o permiso) para llevar a cabo cada tarea asignada a una función.

WordPress viene con seis roles integrados:

Super Administrador es un usuario que tiene acceso completo a características multisitio: puede crear y administrar los sub-sitios, crear y administrar los usuarios de la red, instalar y quitar temas y plugins y capacitarlos en la red.
Administrador es un usuario que tiene acceso total a las funciones de administración de la instalación regular de WordPress.
Editor es un usuario que puede crear, editar y publicar contenido creado por cualquier usuario.
Autor puede publicar y editar sus propios contenidos.
Colaborador puede crear y editar pero no publicar su propio contenido.
Suscriptor sólo puede acceder a su perfil.

Estos roles de usuario de WordPress se ordenan jerárquicamente, lo que significa que las funciones de nivel superior tienen las mismas capacidades de funciones inferiores, además de una serie de capacidades específicas de la función.
Considere la siguiente tabla:

Subscriber Contributor Editor

read	read	read
–	edit_posts	edit_posts
–	delete_posts	delete_posts
–	–	delete_published_posts
–	–	publish_posts
–	–	upload_files
–	–	edit_published_posts

Un Contribuyente tiene la misma capacidad de lectura como un suscriptor, pero él tiene dos capacidades más, editar_posts y borrar_posts, que son específicos para esta función: el contribuyente puede crear, editar y eliminar sus propios posts no publicados.
Un Editor tiene las mismas capacidades como un colaborador, y además él puede publicar, editar y borrar sus propios mensajes, y cargar archivos multimedia.

Tenemos que hacer una distinción entre dos tipos de capacidades:

Meta capacidades dependen del contexto (es decir, el usuario que ha iniciado sesión y el post/página actual).
Funciones Primitivas no dependen del contexto (es decir, editar posts creados por otros usuarios)

Un usuario puede editar un post si él es el autor del post (meta capacidad 'edit_post') o si él tiene la capacidad de editar los posts creados por otros usuarios (Capacidad primitiva'edit_other_posts').
WordPress automáticamente traduce meta capacidades en una o más funciones primitivas para puestos ordinarios, pero cuando se utilizan tipos de post tenemos que asignar manualmente meta capacidades a capacidades primitivas.
Vea el Codex para obtener una lista completa de las funciones integradas.

Los roles y capacidades predeterminados suelen bastar para los requisitos más comunes de un sitio web, pero a veces se le pedirá que proporcione el admin de sitio con un control más granular sobre lo que los usuarios pueden ver y hacer.
Afortunadamente, los usuarios de WordPress y/o desarrolladores, no estamos limitados a funciones y capacidades predeterminadas, porque WordPress nos permite crear nuevas capacidades y funciones, y asignar diferentes conjuntos de capacidades a las funciones existentes. Por ejemplo, puede asignar a los suscriptores la capacidad de crear y editar posts y/o asignar a los contribuyentes la posibilidad de publicar.
Pero usted puede conseguir mucho más del sistema de gestión de usuario:

Puede mostrar/ocultar elementos front-end – como elementos de menú, puestos o widgets – dependiendo del rol de usuario
Puede personalizar el contenido del post por usuario
Puede restringir el acceso a determinadas secciones del sitio para usuarios o roles específicos
Puede crear tipos de post personalizados con funciones personalizadas que pueden ser disminuidas de manera diferente para cada función

Este último punto marca el objetivo de este post.

Estudiantes, Profesores y Tarea

Supongamos que está creando un sitio web educativo donde los estudiantes y profesores pueden interactuar unos con otros. Los estudiantes deben crear proyectos y enviarlos para su revisión.
Los profesores deben ser capaces de editar proyectos estudiantiles y publicarlos si se aprueban.
Se permiten comentarios: Los estudiantes pueden comentar cualquier proyecto, pero sólo los maestros pueden moderar los comentarios.
Aquí está nuestra lista de tareas pendientes:

Agregar roles de usuario de estudiante y maestro

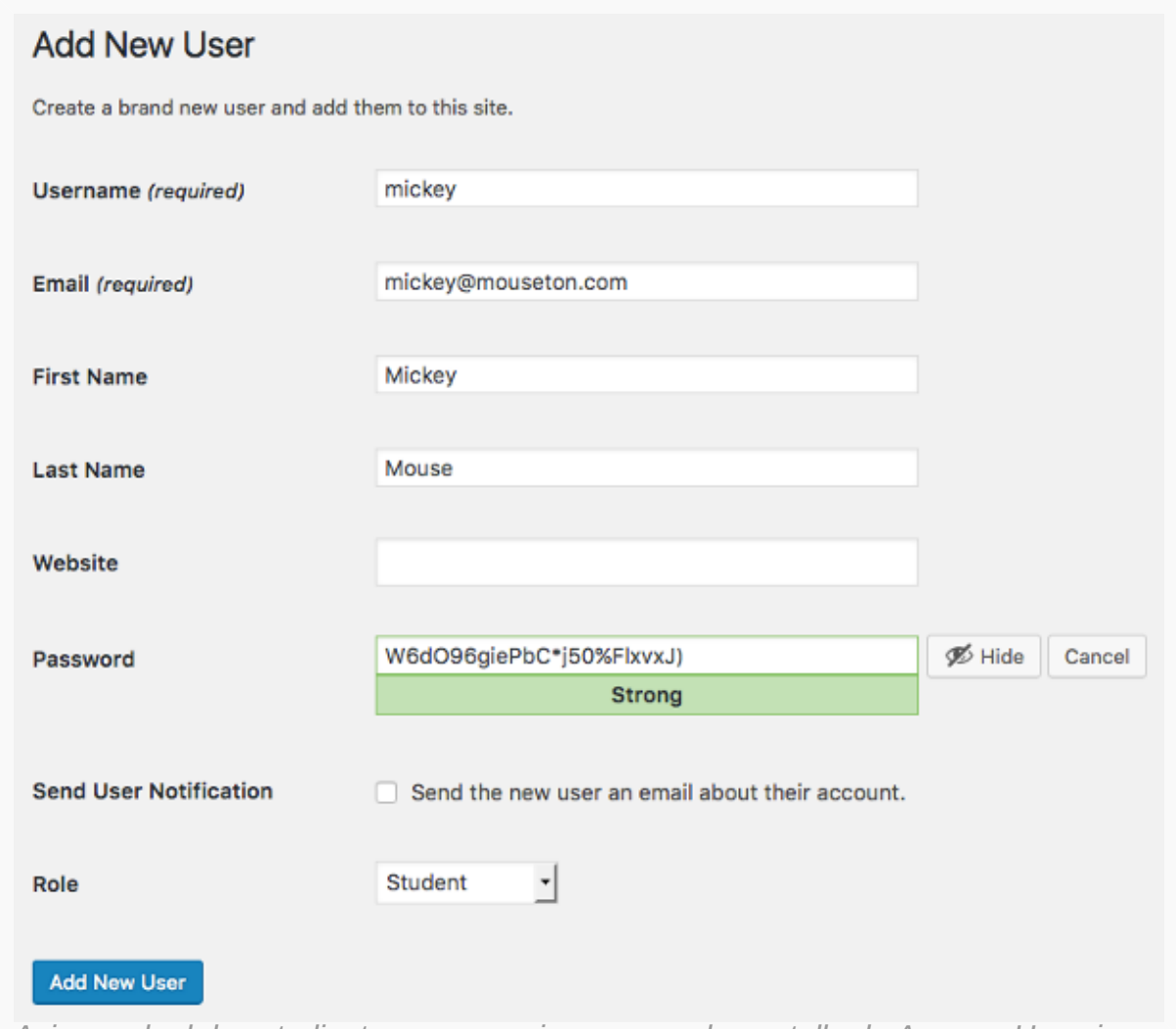
Registrar el tipo de post de estudiante_proyecto y la taxonomía personalizada del sujeto
Registrar y asignar funciones específicas para el tipo de post de estudiante_proyecto
Asignar capacidades a roles de administrador, estudiante y profesor

Vamos a diseccionar el tema principalmente desde la perspectiva de un desarrollador, construyendo un plugin que registra funciones y capacidades Tipo de Post Personalizado y taxonomía. Pero no olvidemos a los que no son desarrolladores, así en la parte final de este post voy a sugerir algunos de los plugins más populares y completos que encontrará para gestionar los roles de usuario y capacidades como un profesional sin tener que escribir una sola línea de código.

Antes de leer el artículo, eche un vistazo al código del plugin en Gist.

Registrar Funciones de Usuario y Capacidades de WordPress

Los estudiantes serán permitidos a leer todos los proyectos y crear, editar y eliminar sus propios proyectos. No se les permitirá publicar ningún proyecto, ni editar y eliminar proyectos publicados.
Los profesores deben controlar cada aspecto de la administración del proyecto. Se les permitirá crear nuevos proyectos, editar, borrar y publicar los proyectos creados por cualquier usuario.



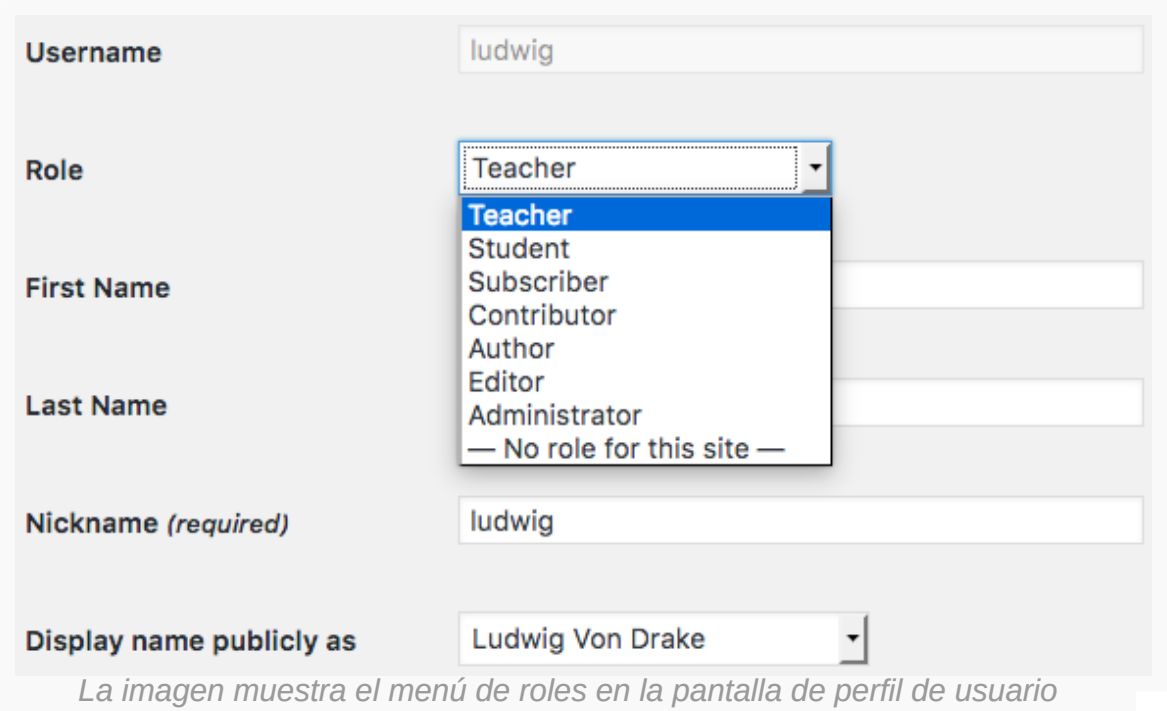
Asignar el rol de estudiante a un usuario nuevo en la pantalla de Agregar Usuarios Nuevos

Dicho esto, vamos a registrar roles de estudiante y profesor:

```
function kinsta_add_roles() {
add_role( 'student', 'Student', array(
'read' => true,
'edit_posts' => true,
'delete_posts' => true ) );
add_role( 'teacher', 'Teacher', array(
'read' => true,
'edit_posts' => true,
'delete_posts' => true,
'delete_published_posts' => true,
'publish_posts' => true,
'upload_files' => true,
'edit_published_posts' => true,
'manage_categories' => true ) );
}
register_activation_hook( __FILE__, 'kinsta_add_roles' );
```

register_activation_hook registra una función que se ejecuta cuando un plugin está activado. Aquí es preferible un hook de acción como init, porque los nuevos papeles están registrados en la base de datos, y no necesitamos ejecutar la función siempre cuando WordPress se
cargue. register_activation_hook mantiene dos argumentos: la ruta de acceso al archivo principal del plugin (__FILE__), y la función de devolución de llamada a ejecutarse ('kinsta_add_roles').
Sobre la activación del plugin,register_activation_hook es ejecutado, y add_role registra el nuevo rol en la tabla wp_options si no existe. Esta segunda función mantiene tres argumentos: el nombre de la función, el nombre para mostrar y una variedad de capacidades.

La función de devolución de llamada agrega dos funciones. Cada función se suministra con un conjunto de capacidades diferentes: los estudiantes podrán leer los posts públicos, crear, editar y borrar sus propios posts si no se han publicado. Los profesores serán autorizados a publicar sus propios posts, editar y borrar posts publicados, cargar archivos multimedia y administrar categorías.



La imagen muestra el menú de roles en la pantalla de perfil de usuario

Ahora podemos crear una comunidad de estudiantes y profesores, pero no son mucho más que los contribuyentes y los editores. Por lo tanto, vamos un paso adelante, y registrar un Custom Post Type para estudiantes, profesores y administradores del sitio.

Registrar un Tipo de Post Personalizado y Taxonomía Relacionada

El siguiente código registra el tipo de post de estudiante_proyecto:

```
function kinsta_project_post_type(){

// define an array of labels
$post_type_labels = array(
'name' => __( 'Projects' ),
'singular_name' => __( 'Project' ),
'add_new_item' => __( 'Add New Project' ),
'edit_item' => __( 'Edit Project' ),
'new_item' => __( 'New Project' ),
'view_item' => __( 'View Project' ),
'view_items' => __( 'View Projects' ),
'not_found' => __( 'No Projects found' ),
'not_found_in_trash' => __( 'No Projects found in Thrash' ),
'all_items' => __( 'All Projects' ),
'archives' => __( 'Project Archives' ),
'insert_into_item' => __( 'Insert into Project' ),
'uploaded_to_this_item' => __( 'Uploaded to this Project' )
);

// define an array of arguments
$post_type_args = array(
'labels' => $post_type_labels,
'public' => true,
'menu_position' => 5,
'menu_icon' => 'dashicons-media-document',
'hierarchical' => false,
'supports' => array( 'title', 'editor', 'author', 'excerpt', 'custom-fields',
'comments' ),
'taxonomies' => array( 'subject' ),
'has_archive' => true,
);

register_post_type( 'student_project', $post_type_args );

}
add_action( 'init', 'kinsta_project_post_type' );
```

A fin de registrar un nuevo tipo de post, necesitamos llamar a la función register_post_type, que tiene que ser invocada mediante la acción init.
Register_post_type mantiene dos argumentos: el slug del tipo de post (“estudiante_proyecto”), y un conjunto de argumentos que configuran las etiquetas admin y parámetros de tipo de post.
Después, vamos a registrar una taxonomía no jerárquica que identifica el tema del proyecto:

```
function kinsta_project_post_type(){

$taxonomy_labels = array(
'name' => __( 'Subjects' ),
'singular_name' => __( 'Subject' ),
'search_items' => __( 'Search Subjects' ),
'popular_items' => __( 'Popular Subjects' ),
```

```
'all_items' => __( 'All Subjects' ),
'parent_item' => null,
'parent_item_colon' => null,
'edit_item' => __( 'Edit Subject' ),
'update_item' => __( 'Update Subject' ),
'add_new_item' => __( 'Add New Subject' ),
'new_item_name' => __( 'New Subject Name' ),
'separate_items_with_commas' => __( 'Separate subjects with commas' ),
'add_or_remove_items' => __( 'Add or remove subjects' ),
'choose_from_most_used' => __( 'Choose from the most used subjects' ),
'not_found' => __( 'No subjects found.' ),
'menu_name' => __( 'Subjects' ),
);

$taxonomy_args = array(
'hierarchical' => false,
'labels' => $taxonomy_labels,
'show_ui' => true,
'show_admin_column' => true,
'update_count_callback' => '_update_post_term_count',
'query_var' => true,
'rewrite' => array( 'slug' => 'subject' ),
);

register_taxonomy( 'subject', 'student_project', $taxonomy_args );
}
add_action( 'init', 'kinsta_project_post_type' );
```

register_taxonomy mantiene tres argumentos: el slug de la taxonomía (‘subject’), el tipo de post vinculado (‘estudiante_proyecto’), una matriz de argumentos almacenando etiquetas y otros parámetros (véase el Codex para más información).
Al igual que register_post_type, register_taxonomy debe estar enganchado a la acción init.
Una vez hecho esto, es hora de definir el conjunto de capacidades que permiten los diferentes niveles de control sobre el tipo de post.

Un Conjunto de Capacidades Específicas para Tipo de Post de Proyecto del Estudiante

A fin de añadir funcionalidades específicas para el nuevo tipo de post, deberíamos utilizar uno o dos de los siguientes argumentos de register_post_type:

capability _type: (string|array) una cadena a ser usada como base para construir las capacidades leer, editar y eliminar (ejemplo.‘student _project’). Si necesita pasar plurales alternativos (es decir, historia/historias), debe establecer su valor a un array (p. ej. array(‘story”, ‘stories”
). capability_type genera las siguientes capacidades:

Meta capacidades:

- edit_{capability_type}
- read_ {capability_type}
- delete_{capability_type}

Capacidades Primitivas:

- edit_{capability_type}s
- edit_others_{capability_type}s

- publish_{capability_type}s
- read_private_{capability_type}s

Funciones Primitivas definidas en la función mapa_meta_cap:

- read
- delete_{capability_type}s
- delete_private_{capability_type}s
- delete_published_{capability_type}s
- delete_others_{capability_type}s
- edit_private_{capability_type}s
- edit_published_{capability_type}s

capability_type otorga un control general sobre las capacidades de tipo de post. Si necesita un control más granular, puede utilizar el argumento de capacidades.

capabilities: (Matriz) es un conjunto de capacidades para el tipo de post. Si elige este argumento en lugar de capability_type, es posible que lo pase a la función de register_post_type de la siguiente manera:

```
'capabilities' => array(
'read_post' => 'read_student_project',
'read_private_posts' => 'read_private_student_projects',
'edit_post' => 'edit_student_project',
'edit_posts' => 'edit_student_projects',
'edit_others_posts' => 'edit_others_student_projects',
'edit_published_posts' => 'edit_published_student_projects',
'edit_private_posts' => 'edit_private_student_projects',
'delete_post' => 'delete_student_project',
'delete_posts' => 'delete_student_projects',
'delete_others_posts' => 'delete_others_student_projects',
'delete_published_posts' => 'delete_published_student_projects',
'delete_private_posts' => 'delete_private_student_projects',
'publish_posts' => 'publish_student_projects',
'moderate_comments' => 'moderate_student_project_comments',
),
```

map_meta_cap: (boolean) convierte automáticamente las meta capacidades en una o más funciones primitivas para el tipo de post. En nuestro ejemplo, se debe establecer en true cuando se utiliza el argumento de capacidades, pero podría ser necesario ajustar su valor a false cuando se utiliza el argumento capability_type(más sobre este tema en el Codex).

Ahora podemos volver a nuestra función register_post_type y definir una nueva matriz de argumentos como sigue:

```
$post_type_args = array(
```

```
'labels' => $post_type_labels,
'public' => true,
'menu_position' => 5,
'menu_icon' => 'dashicons-media-document',
//'capability_type' => 'student_project',
'capabilities' => array(
'read_post' => 'read_student_project',
'read_private_posts' => 'read_private_student_projects',
'edit_post' => 'edit_student_project',
'edit_posts' => 'edit_student_projects',
```

```
'edit_others_posts' => 'edit_others_student_projects',
'edit_published_posts' => 'edit_published_student_projects',
'edit_private_posts' => 'edit_private_student_projects',
'delete_post' => 'delete_student_project',
'delete_posts' => 'delete_student_projects',
'delete_others_posts' => 'delete_others_student_projects',
'delete_published_posts' => 'delete_published_student_projects',
'delete_private_posts' => 'delete_private_student_projects',
'publish_posts' => 'publish_student_projects',
'moderate_comments' => 'moderate_student_project_comments',
),
'map_meta_cap' => true,
'hierarchical' => false,
'supports' => array( 'title', 'editor', 'author', 'excerpt', 'custom-fields',
'comments' ),
'taxonomies' => array( 'subject' ),
'has_archive' => true,
);

register_post_type( 'student_project', $post_type_args );
```

Las capacidades están totalmente documentados en el archivo /wp-includes/post.php y en referencia de función register_post_type.

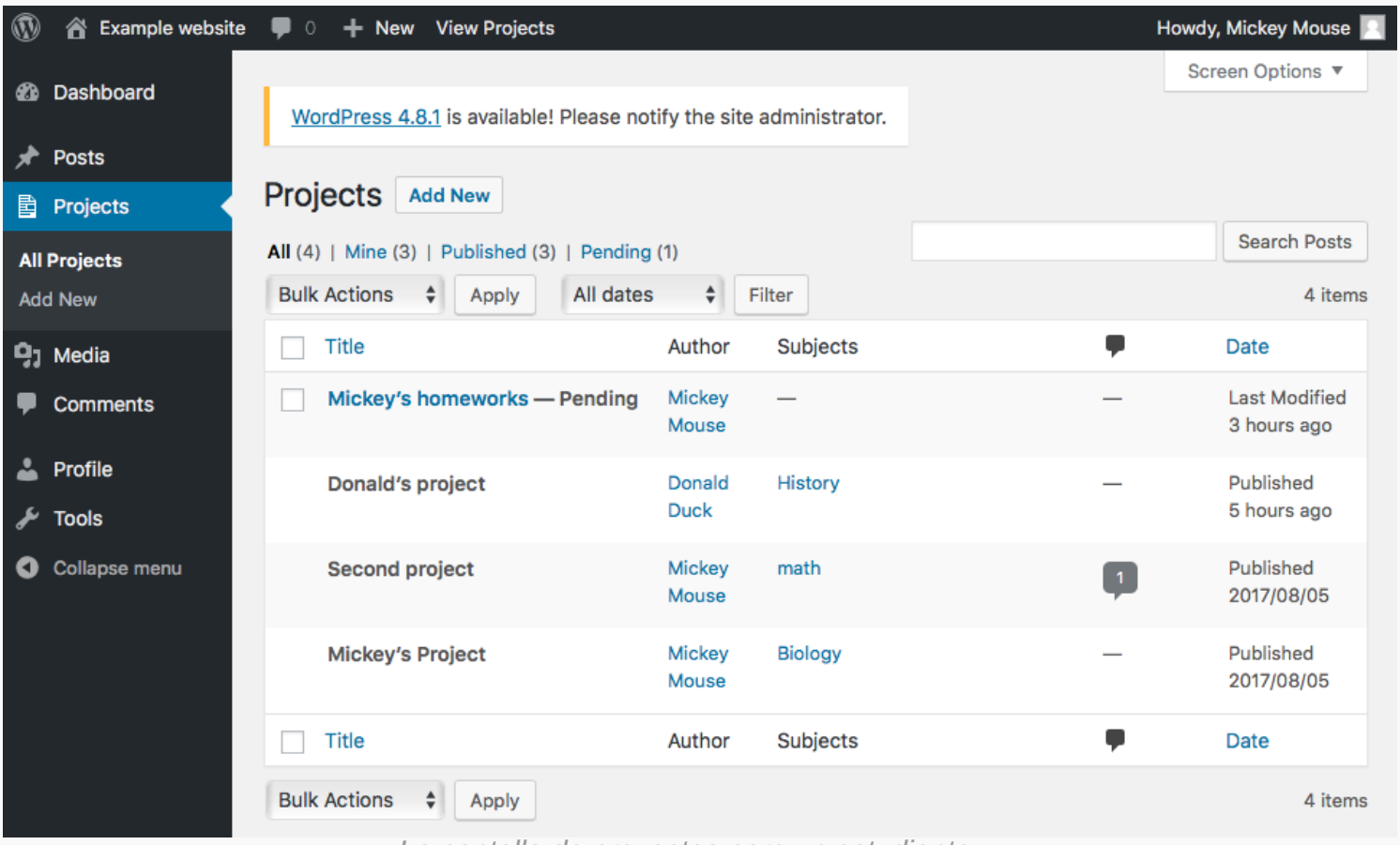
Añadiendo Capacidades a Profesores y Estudiantes

Ahora que hemos registrado dos funciones y un tipo de post con capacidades específicas, nuestra tarea es asignar funciones a roles:

```
function kinsta_add_caps(){
$admin = get_role( 'administrator' );
$admin->add_cap( 'read_student_project' );
$admin->add_cap( 'read_private_student_project' );
$admin->add_cap( 'edit_student_project' );
$admin->add_cap( 'edit_student_projects' );
$admin->add_cap( 'edit_others_student_projects' );
$admin->add_cap( 'edit_published_student_projects' );
$admin->add_cap( 'edit_private_student_projects' );
$admin->add_cap( 'delete_student_projects' );
$admin->add_cap( 'delete_student_project' );
$admin->add_cap( 'delete_others_student_projects' );
$admin->add_cap( 'delete_published_student_project' );
$admin->add_cap( 'delete_student_project' );
$admin->add_cap( 'delete_private_student_project' );
$admin->add_cap( 'publish_student_projects' );
$admin->add_cap( 'moderate_student_project_comments' );

$student = get_role( 'student' );
$student->add_cap( 'read_student_project' );
$student->add_cap( 'edit_student_project' );
$student->add_cap( 'edit_student_projects' );
$student->add_cap( 'delete_student_project' );
$student->add_cap( 'delete_student_projects' );

$teacher = get_role( 'teacher' );
$teacher->add_cap( 'read_student_project' );
$teacher->add_cap( 'read_private_student_project' );
$teacher->add_cap( 'edit_student_project' );
$teacher->add_cap( 'edit_student_projects' );
$teacher->add_cap( 'edit_others_student_projects' );
$teacher->add_cap( 'edit_published_student_projects' );
$teacher->add_cap( 'edit_private_student_projects' );
$teacher->add_cap( 'delete_student_project' );
$teacher->add_cap( 'delete_student_projects' );
$teacher->add_cap( 'delete_others_student_projects' );
$teacher->add_cap( 'delete_published_student_projects' );
$teacher->add_cap( 'delete_private_student_project' );
```

```
$teacher->add_cap( 'publish_student_projects' );
$teacher->add_cap( 'moderate_student_project_comments' );
}
add_action( 'admin_init', 'kinsta_add_caps');
```

Este código es bastante autoexplicativo: la función de devolución de llamada está enganchada a la acción admin_init, que dispara antes que cualquier otro hook cuando el panel de admin está cargado. El método add_cap del objeto WP_Role asigna la capacidad especificada a un rol.



La pantalla de proyectos para un estudiante

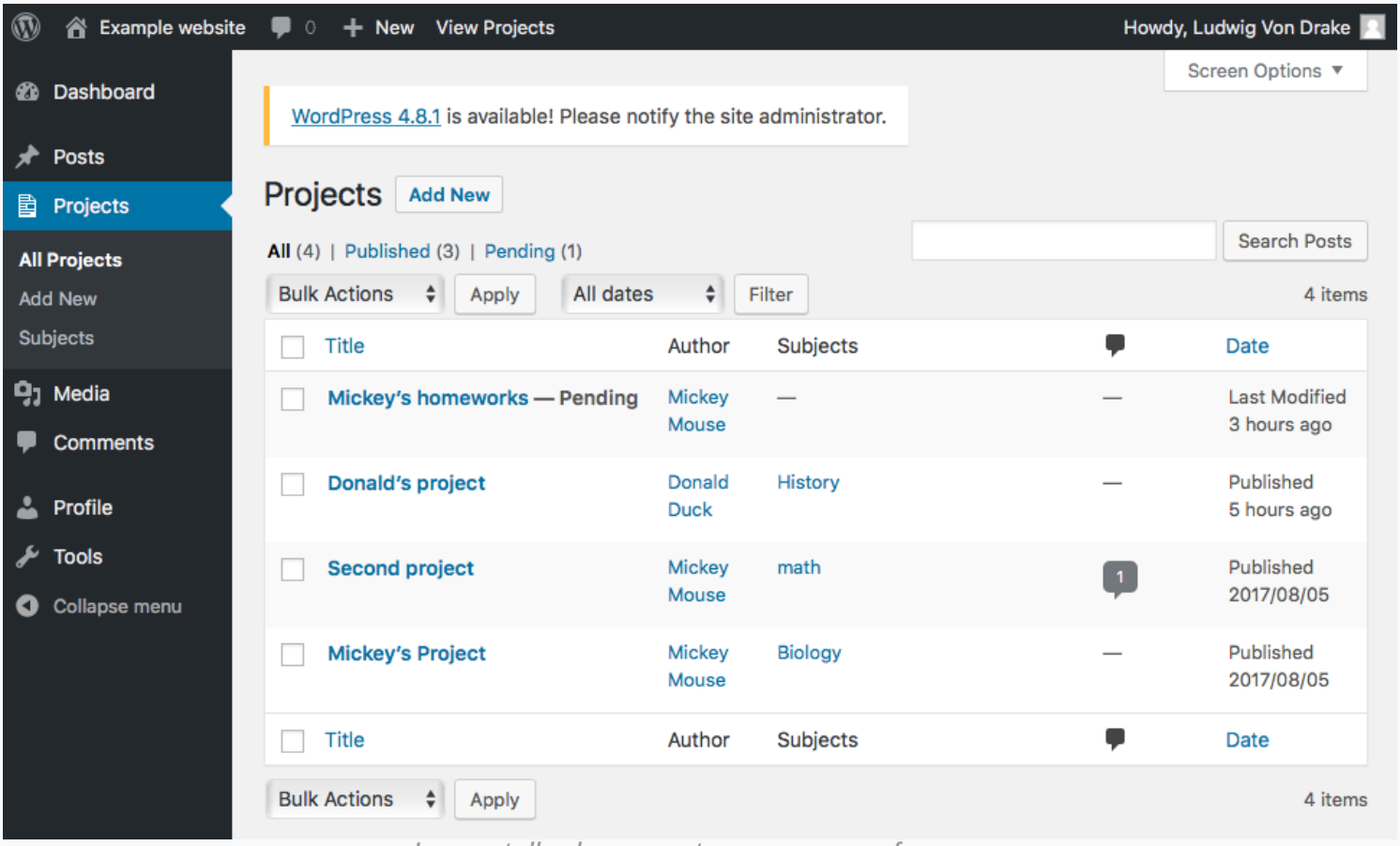
Limpieza de la Base de Datos sobre Desactivación del Plugin

Por último, podemos eliminar el rol de estudiante y profesor, y eliminar capacidades de tipo de post personalizado del rol de admin en la desactivación del plugin:

```
function kinsta_remove_roles(){
//check if role exist before removing it
if( get_role('student') ){
remove_role( 'student' );
}
if( get_role('teacher') ){
remove_role( 'teacher' );
}

$admin = get_role( 'administrator' );

$caps = array(
'read_student_project',
```

```
'read_private_student_project',
'edit_student_project',
'edit_student_projects',
'edit_others_student_projects',
'edit_published_student_projects',
'edit_private_student_projects',
'delete_student_projects',
'delete_student_project',
'delete_others_student_projects',
'delete_published_student_project',
'delete_student_project',
'delete_private_student_project',
'publish_student_projects',
'moderate_student_project_comments'
);

foreach ( $caps as $cap ) {
$admin->remove_cap( $cap );
}
}
register_deactivation_hook( __FILE__, 'kinsta_remove_roles' );
```

El proyecto ya está completo, se puede agarrar el código completo de los ejemplos de este Gist público.



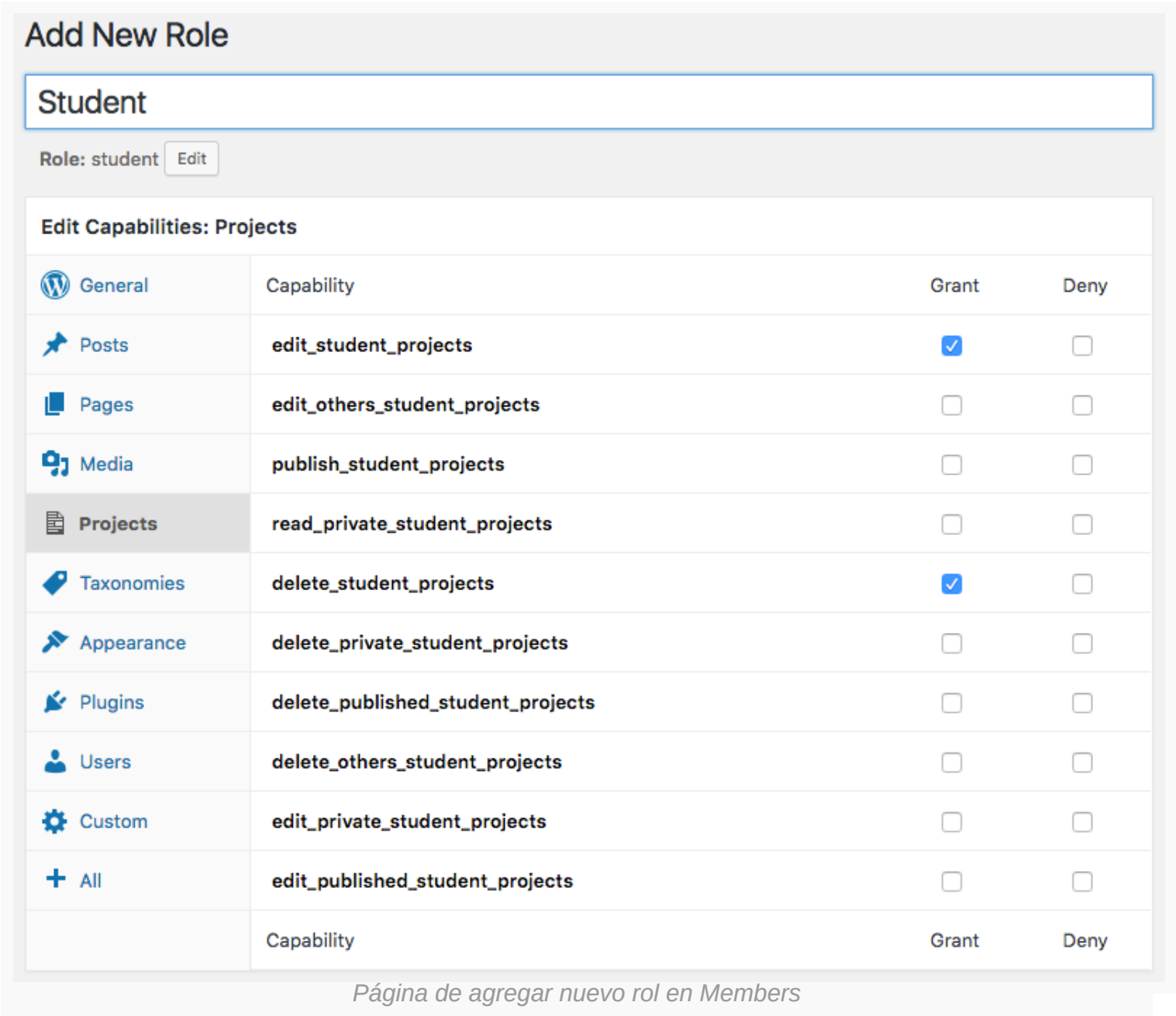
La pantalla de proyectos para un profesor

Control de los Roles de Usuario y Capacidades de WordPress con Plugins

Si no es un programador, puede tomar el control de las funciones y capacidades de todos modos, gracias a una serie de plugins gratuitos disponibles en el directorio de plugins.

Members por Justin Tadlock es un editor de rol y capacidaddx fácil de utilizar y completo. Permite que el

administrador del sitio agregue, edite y quite funciones y capacidades, para asignar a los usuarios más de una capacidad, negar capacidades específicas a determinadas funciones, controlar qué roles pueden acceder a publicar contenido, proporciona los códigos de acceso y los widgets, y más.



Página de agregar nuevo rol en Members

User Role Editor por Vladimir Garagulya es otro plugin que proporciona un control completo sobre las funciones y capacidades. Además, permite a los administradores del sitio asignar roles a usuarios individuales y otorga compatibilidad Multisitio. Este plugin también está disponible en una versión comercial con funcionalidades adicionales.

WPFront User Role Editor es otra opción para gestionar los roles y capacidades en WordPress. Está disponible en una versión gratuita en el directorio de plugins, y en una versión comercial con más funciones avanzadas.

Aunque no es un editor de rol User Switching es un plugin imprescindible para la gestión de rol y capacidad. Su único propósito es permitir que el administrador del sitio intercambie entre las cuentas de usuario con un solo clic, de modo que él no necesite iniciar la sesión y salir de la sesión repetidamente para controlar lo que los usuarios pueden ver en el sitio web.

Conclusiones

A primera vista, los conceptos de rol y capacidad de usuario de WordPress puede ser un poco confuso, especialmente a lo que se refiere a meta capacidades y funciones primitivas. De todos modos, una vez que haya centrado el concepto clave del sistema de gestión de usuarios de WordPress usted puede conseguir fácilmente un control granular sobre las actividades de los usuarios.