pagepro

WordPress Performance Guide

Best practices for WP developers

Server

1. Site health

All the most important information about your site on WordPress and the server configuration you can find on the Tools / Site Health tabs in your WP admin.

Check if you can see critical issues or recommended improvements and try to fix all of them.

2. Use required PHP version

You can always check the minimum WP requirements here:

https://wordpress.org/about/requirements/

At the moment you shouldn't use PHP below 7.4.

3. InnoDB

Make sure that database uses InnoDB tables instead of MyISAM. InnoDB uses transactions and foreign keys, so it's safer and it can handle more complex queries faster than MyISAM.

4. Enable GZIP compression

Enable GZIP compression using W3 Total Cache plugin.

If it's possible, the best way to enable GZIP compression is at the server-level of Apache or Nginx.

Don't GZIP images, PDF or other binary data.

GZIP data only in the range of 150-1000 bytes in size. The speed of compression must be faster than the time taken in delivering the content uncompressed.

Don't compress content for old browsers.

Not following the above recommendations actually increases file size and page load times due to overhead of compression and decompression.

```
AddOutputFilterByType DEFLATE text/plain
AddOutputFilterByType DEFLATE text/html
AddOutputFilterByType DEFLATE text/xml
AddOutputFilterByType DEFLATE text/css
AddOutputFilterByType DEFLATE application/xml
AddOutputFilterByType DEFLATE application/xhtml+xml
AddOutputFilterByType DEFLATE application/rss+xml
AddOutputFilterByType DEFLATE application/javascript
AddOutputFilterByType DEFLATE application/x-javascript
```

5. Memcache

Check if your hosting provider allows you to use memcache. If it's possible, enable this feature on the server and install one of the cache plugins.

W3 Total Cache allows you to configure memcache, so it's one of the options you can use.

6. DNS level website firewall

Don't use Wordfence, Better Security or any other plugin as a firewall. It can block specific users but attackers still can reach the web server before they get blocked.

Use DNS level traffic provided by Cloudflare or Sucuri. These firewalls help you block brute force attacks, hacking attempts and malware.

7. Mailserver

Don't use the same web server as a mailserver. Use SMTP or Mailgun instead.

8. Server cache

Check if your server provides any of the cache on the server side. This type of cache gives noticeable better performance results than cache provided by WordPress plugins.

You can use these types of server caches:

- Litespeed
- Redis
- https://varnish-cache.org/

9. Use PHP-FPM

Usually you can't see a difference between mod_php and PHP-FPM. But they work differently on the server side and PHP-FPM provides much better performance. If it's possible, use PHP-FPM rather than mod_php.

10. Use HTTP/2

HTTP/2 solves some problems which the old HTTP/1.1 had. It makes a difference in performance and HTTP/2 works faster. It uses prioritization and provides better header compression.

Development

1. Keep WP up to date

Always use the latest version of WordPress.

Remember to test your site after updating the test environment.

2. Keep all the plugins up to date

It's important to use the latest versions of all the plugins. Also, updating should be scheduled and done periodically.

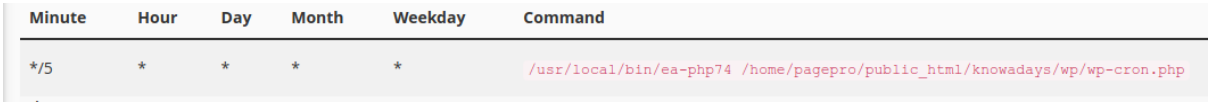
During updating look at the latest release date of each plugin. If any of them didn't have any update for a long time (>1 year), check if it doesn't have any issues on your site.

3. Use server cron jobs

Turn off WP cron system by adding this line into the *wp-config.php* file:

```
define( 'DISABLE_WP_CRON', true );
```

Set the cron job in cpanel:



Minute	Hour	Day	Month	Weekday	Command
*/5	*	*	*	*	/usr/local/bin/ea-php74 /home/pagepro/public_html/knowadays/wp/wp-cron.php

4. Don't *flush_rewrite_rules()* if not needed

Function *flush_rewrite_rules()* starts a very expensive operation. This function should be triggered only once, when it's necessary. It shouldn't be used on each page loading. Be aware that any of the active plugins also shouldn't do that.

Search for *flush_rewrite_rules()* function in the code and decide if any use of this is really required.

5. Use WP_Query instead of get_posts()

get_posts() function calls WP_Query by itself, but WP_Query allows to use more arguments than get_posts(). You should use WP_Query rather than get_posts() because it allows to have better performance with these parameters, which can be overwritten when get_posts() is used:

- *'not_found_rows' => true*

  Use this argument, when pagination is not needed.

- *'update_post_meta_cache' => false*

  Use this argument, when post meta shouldn't be utilized.

- *'update_post_term_cache' => false*

Use this argument, when taxonomy terms shouldn't be utilized.

- *'fields' => 'ids'*

  Use this argument, when it's enough to get post IDs from the query.

6. Don't use uncacheable core functions

Some of the core WordPress functions are uncached. It means that when one of these functions is called, it always runs a new SQL query.

If it's possible, try to not use functions listed below:

- *attachment_url_to_postid()*
- *count_user_posts()*
- *get_adjacent_post()*
- *get_previous_post()*
- *get_next_post()*
- *previous_post_link()*
- *next_post_link()*
- *get_children()*
- *get_posts()*
- *get_page_by_path()*
- *get_page_by_title()*
- *term_exists()*
- *url_to_postid()*
- *wp_get_post_terms()*
- *wp_get_recent_posts()*
- *wp_oembed_get()*
- *wp_old_slug_redirect()*

7. Don’t use post__not_in

Additional “*post__not_in*” parameter in the WP_Query can make the query slower. But also it can make the same query different for each “*post__not_in*” value. Because of that WP won’t use cached query but it will create a new query each time.

Instead of “*post__not_in*” argument, get all the posts you need, and exclude specific posts by for each loop in PHP.

8. Exclude taxonomy children from the query

Add ‘*include_children*’ => false into the taxonomy query. It makes taxonomy query faster.

9. Use wp_remote_get() for remote data

Use *wp_remote_get()* instead of cURL for remote requests and cache the results.

Set timeout (3s) for the remote requests. It can prevent server overloading if the remote server doesn’t respond.

10. Minimize the quantity of autoload records in wp_options table

If a record in the wp_options table has an “autoload” value set to “yes”, this record is loaded on every page of the site.

Check the number of autoloaded values on your site:

```
SELECT SUM(LENGTH(option_value)) as autoload_size FROM
wp_options WHERE autoload='yes';
```

If you can see that your WordPress loads more than 1 MB, you should find the reason why so many options are auto loaded and try to decrease this size.

You can use this query to see the number of the rows in the table and to get the first 10 rows by size:

```
SELECT 'autoloaded data in KiB' as name,
ROUND(SUM(LENGTH(option_value))/ 1024) as value FROM
wp_options WHERE autoload='yes'
UNION
SELECT 'autoloaded data count', count(*) FROM wp_options WHERE
autoload='yes'
UNION
(SELECT option_name, length(option_value) FROM wp_options WHERE
autoload='yes' ORDER BY length(option_value) DESC LIMIT 10)
```

What kind of records can you find?

- Data stored by plugins, eg. contact forms. This data is not required on each page, so it doesn't have to be auto loaded.
- Old, useless data, like settings of the old plugins, which have already been removed.
- Sessions data.
- Transients.

You can also use some plugin to clean the wp_options table.

Remember to make a backup before your work!

Also, schedule the same process and do it periodically to keep wp_options table clean all the time.

11. Add MySQL index to options table

12. Remove emojis

```
/**
* Disable emoji
*/
function {theme_prefix}_disable_emojis() {
  remove_action( 'wp_head', 'print_emoji_detection_script', 7 );
  remove_action( 'admin_print_scripts', 'print_emoji_detection_script' );
  remove_action( 'wp_print_styles', 'print_emoji_styles' );
  remove_action( 'admin_print_styles', 'print_emoji_styles' );
  remove_filter( 'the_content_feed', 'wp_staticize_emoji' );
  remove_filter( 'comment_text_rss', 'wp_staticize_emoji' );
  remove_filter( 'wp_mail', 'wp_staticize_emoji_for_email' );
  add_filter( 'tiny_mce_plugins', '{theme_prefix}_disable_emojis_tinymce'
);
  add_filter( 'wp_resource_hints',
'{theme_prefix}_disable_emojis_remove_dns_prefetch', 10, 2 );
}
add_action( 'init', '{theme_prefix}_disable_emojis' );

/**
* Filter function used to remove the tinymce emoji plugin.
*
* @param array $plugins
* @return array Difference between the two arrays
*/
function {theme_prefix}_disable_emojis_tinymce( $plugins ) {
  if ( is_array( $plugins ) ) {
    return array_diff( $plugins, array( 'wpemoji' ) );
    } else {
  return array();
  }
}
```

```
/**
* Remove emoji CDN hostname from DNS prefetching hints.
*
* @param array $urls URLs to print for resource hints.
* @param string $relation_type The relation type the URLs are printed
for.
* @return array Difference between the two arrays.
*/


function {theme_prefix}_disable_emojis_remove_dns_prefetch( $urls,
$relation_type ) {
  if ( 'dns-prefetch' == $relation_type ) {
    /** This filter is documented in wp-includes/formatting.php */
    $emoji_svg_url = apply_filters( 'emoji_svg_url',
'https://s.w.org/images/core/emoji/2/svg/' );

    $urls = array_diff( $urls, array( $emoji_svg_url ) );
  }

  return $urls;
}
```

13. Remove useless plugins

It's not about performance but it's more secure to remove all the plugins which are not necessary. Don't make a mess in your plugin's list and keep it clean.

14. Replace small, simple plugins by custom theme functions

15. Split long posts into pages

Long content of the post can take too long to load. WordPress provides *<!--nextpage-->* to split them into multiple pages.

16. Limit post revisions

```
define( 'WP_POST_REVISIONS', 3 );
```

17. Use cache for AJAX

If it's possible, try to cache ajax requests for all users.

Use rewrite rules to change admin-ajax.php requests with parameters to clean URLs that can be easily cached, like:

http://example.com/ajax/my_frontend_ajax_function/parameter_1/

18. Fix HTTPS/SSL errors without plugins

If a "Mixed Content" message is visible in the browser's console, don't fix it by using the "Really Simple SSL" plugin or by any similar plugin. Plugins usually change URLs all the time before the content is displayed for the user.

Instead of making the same operation multiple times, just fix all the URLs manually or by "Search & Replace" feature, and make sure that all the URLs are permanently fixed.

19. Remove spam comments

Don't keep garbage data like spam and unapproved comments. Remove them regularly.

20. Purge trash automatically

```
define( 'EMPTY_TRASH_DAYS', 3 );
```

21. Enable cache plugin

Recommended: W3 Total Cache

or Litespeed Cache plugin if Litespeed is provided by the hosting provider.

- Page Caching
- Minification
- Database Caching
- Headers
- CDN
- Browser Caching

22. Avoid redirects

Make sure that users are redirected to the correct URL directly, with no additional redirection, eg,

WRONG : http://domain.com -> http://www.domain.com -> https://www.domain.com

GOOD : http://domain.com -> https://www.domain.com (301)

23. Use minified core scripts

If core JS files have to be used (eg. jquery), make sure that you use minified versions of these files (jquery.min.js instead of jquery.js)

24. Preload components

25. Post-load components

26. Use GET instead of POST for AJAX requests

27. Use GTM for scripts

If your scripts are not critical for the website, consider moving them to the Google Tag Manager. Scripts added by GTM don't trigger "render blocking" alert in Google Page Speed tool.
You can use GTM for analytics and also for 3rd party scripts like widgets or chats.

28. Make favicon small and cacheable

Favicon seems to be not necessary but browsers and a lot of bots request it. It's better not to respond with a 404. Make favicon as small as possible and set expires header:

```
<FilesMatch "\.(jpg|jpeg|png|gif|ico)$">
Header set Cache-Control "max-age=2419200, public"
</FilesMatch>
```

29. Disable trackbacks and pingback

Dashboard / Settings / Discussion

30. Multiple hostnames

https://wordpress.org/support/article/optimization/#multiple-hostnames

http://www.larre.com/2010/01/24/amazon-s3-and-cloudfront-with-wordpress-and-dreamhost/

Front End

1. Do not use @import

2. Remove unused CSS

3. Don’t use CSS in HTML tags

4. Use small inline CSS

5. Use CSS sprite

6. Minify JavaScript and CSS

7. Reduce the number of DOM elements

Images

1. Lazy loading

Consider lazy-loading offscreen and hidden images after all critical resources have finished loading to lower time to interactive

https://wordpress.org/plugins/search/lazy+load/

or use <img loading="lazy">

Optimized images load faster and consume cellular data.

2. Image CDNs

https://web.dev/image-cdns/

3. Image compression

https://web.dev/use-imagemin-to-compress-images/

4. Animated GIFs replaced by video

https://web.dev/replace-gifs-with-videos/

5. Embedded videos instead of self-hosted

Don’t waste the bandwidth of the web hosting. If it’s possible, use 3rd party services for video and audio files. You can use YouTube, Vimeo, DailyMotion, SoundCloud or any other service and use their bandwidth. Also, it’s easier to restore the website from backup if it doesn’t contain large files.

6. Responsive images

https://web.dev/serve-responsive-images/

7. Correct dimension

https://web.dev/serve-images-with-correct-dimensions/

- Add custom sizes by add_image_size()
- Don’t use “full” size of image
- Don’t use [‘url’] of the ACF image object

8. Webp images

https://web.dev/serve-images-webp/

9. Avoid empty image src

If the website contains an image with an empty string src, the browser makes another request to the server. Depending on the browser, it can be requested to the directory or to the current page.

JavaScript

1. Enqueue JS and CSS

Don't add CSS and JS files directly in the head section of the html. WordPress provides functions for this:

- wp_enqueue_script()
- wp_enqueue_style()

2. Minify & combine

Use Autoptimize for minifying and combining JS files.

Combining is required only if HTTP/1.1 is used. HTTP/2 protocol has improved connection concurrency for requests.

3. Reduce external HTTP requests

4. Remove useless JS and CSS files

5. Use required assets on specific pages only

Check this plugin: https://wordpress.org/plugins/wp-asset-clean-up/

ACF

1. Use ACF JSON configuration files

```
/**
* Save ACF fields to JSON files.
*
* @param string $path
* @return void
*/
function {theme_prefix}_acf_json_save_point( $path ) {
  $path = get_stylesheet_directory() . '/inc/acf-json';
  return $path;
}
add_filter( 'acf/settings/save_json', '{theme_prefix}_acf_json_save_point' );



/**
* Load ACF fields from JSON files.
*
* @param array $paths
* @return void
*/
```

```
function {theme_prefix}_acf_json_load_point( $paths ) {

  $paths[] = get_template_directory() . '/inc/acf-json';

  return $paths;

}

add_filter( 'acf/settings/load_json', '{theme_prefix}_acf_json_load_point' );
```

2. Disable ACF on FE

If it's possible, try to disable the ACF plugin on the FE of the website.

It seems to be not possible if Gutenberg editor is in use. But if not, you can replace *get_field()* function with *get_post_meta()*.

We can use a custom function:

```
/**

* Get ACF fields as custom fields

*

* @param [type] $post_id

* @param array $config

* @return void

*/

function {theme_prefix}_get_all_acf( $post_id, array $config ) {



  $results = array();
```

```php
foreach ( $config as $field ) {


  if ( empty( $field['name'] ) ) {

    continue;

  }


  $meta_key = $field['name'];


  if ( isset( $field['meta_key_prefix'] ) ) {

    $meta_key = $field['meta_key_prefix'] . $meta_key;

  }

  $field_value = get_post_meta( $post_id, $meta_key, true );


  if ( isset( $field['layouts'] ) ) { // We're dealing with flexible content layouts.


    if ( empty( $field_value ) ) {

      continue;

    }


    // Build a keyed array of possible layout types.

    $layout_types = [];

    foreach ( $field['layouts'] as $key => $layout_type ) {

      $layout_types[ $layout_type['name'] ] = $layout_type;
```

```
        }

        foreach ( $field_value as $key => $current_layout_type ) {
            $new_config = $layout_types[ $current_layout_type ]['sub_fields'];

            if ( empty( $new_config ) ) {
                continue;
            }

            foreach ( $new_config as &$field_config ) {
                $field_config['meta_key_prefix'] = $meta_key . "_{$key}_";
            }

            $results[ $field['name'] ][] = array_merge(
                [
                    'acf_fc_layout' => $current_layout_type,
                ],
                {theme_prefix}_get_all_acf( $post_id, $new_config )
            );
        }
    } elseif ( isset( $field['sub_fields'] ) && $field['type'] != 'group' ) { // We're dealing with repeater fields.

        if ( empty( $field_value ) ) {
```

```
        continue;
    }

    for ( $i = 0; $i < $field_value; $i ++ ) {
        $new_config = $field['sub_fields'];

        if ( empty( $new_config ) ) {
            continue;
        }

        foreach ( $new_config as &$field_config ) {
            $field_config['meta_key_prefix'] = $meta_key . "_{$i}_";
        }

        $results[ $field['name'] ][] = {theme_prefix}_get_all_acf( $post_id, $new_config );
    }
} elseif ( isset( $field['type'] ) && $field['type'] == 'group' ) { // Group field type
    if ( isset( $field['sub_fields'] ) && $field['sub_fields'] ) {
        $i = 0;
        foreach ( $field['sub_fields'] as $sub_field ) {
            $new_config[$i] = $field['sub_fields'];
            $new_config[$i]['name'] = $field['name'] . '_' .$sub_field['name'];
            $i++;
```

```
        }

        $results[ $field['name'] ] = {theme_prefix}_get_all_acf( $post_id, $new_config );

        foreach ( $results[$field['name']] as $key => $value ) {

          $new_key = str_replace( $field['name'] . '_', '', $key );

          $new_results[$new_key] = $value;

        }

        $results[$field['name']] = $new_results;

      }

    } else {

      $results[ $field['name'] ] = $field_value;

    }

  }

  return $results;

}
```

It can be used on the page template to get all ACF fields of the specific group:

```
$field_group_array = json_decode( file_get_contents( WP_CONTENT_DIR .
"/themes/{theme_name}/inc/acf-json/group_5cf56df487e6c.json" ), true );

$config = $field_group_array['fields'];

$meta_data = {theme_name}_get_all_acf( get_the_ID(), $config );
```

ACF plugin can be disabled on FE by adding custom plugin into the mu-plugins folder:

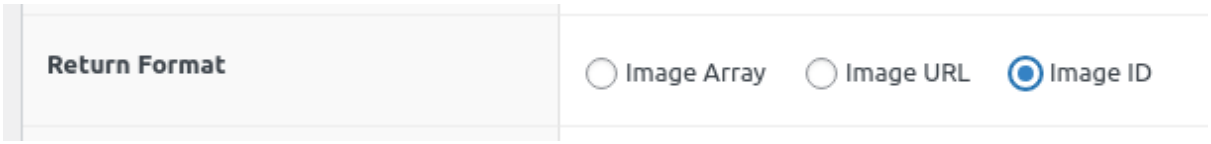
```php
<?php

/**

* @package active-plugins

* @version 1.0

*

* Plugin Name: Disable plugins on FE

* Description: Exclude some plugins on FE

* Author: Karol Sawka

* Version: 1.0

*/



global $request_uri;

$request_uri = parse_url( $_SERVER['REQUEST_URI'], PHP_URL_PATH );

$is_admin = strpos( $request_uri, 'wp-admin/' );



if( !$is_admin ){



  $excluded = array();



  add_filter(

    'option_active_plugins',

    function( $plugins ) {
```

```php
$request_uri = parse_url( $_SERVER['REQUEST_URI'], PHP_URL_PATH );

$excluded = array();



// disable plugins on home page

if ( $request_uri == '/' ) {

   $excluded = array(

      'advanced-custom-fields-pro/acf.php',

      'acfml/wpml-acf.php',

      'acf-code-field/acf-code-field.php',

      'acf-woocommerce-attribute/acf-attribute.php'

   );

}



if( $excluded ){

   foreach ( $plugins as $plugin ) {

      if ( !in_array( $plugin, $excluded ) ) {

         $new_plugins[] = $plugin;

      }

   }

} else {

   $new_plugins = $plugins;

}
```

```
    return $new_plugins;


  }
);


}
```

3. Don’t use an array of the ACF image field

ACF image field has three options:



“Image ID” is the most appropriate for performance. It prevents making useless db queries and allows you to get only the correct image size.

You can get the correct image data by its ID:

```
function {theme_prefix}_get_image_src( $image_id, $size ) {


 $image_url = '';


 if ( $image_id ) {
```

```
    $image_object = wp_get_attachment_image_src( $image_id, $size );
    $image_url = isset( $image_object[0] ) ? $image_object[0] : $image_url;
  }

  return $image_url;
}
```

```
$image_id = get_field( 'image' );
$image_url = {theme_prefix}_get_image_src( $image_id, 'hero' );
```

4. Use taxonomies for searchable fields

Post meta queries are more expensive than taxonomy terms.

WPML

1. Memory limit - 256 MB

WordPress allocates 32 MB memory of PHP by default. It’s not enough for a multilingual website. Make sure that it’s increased to 256 MB.

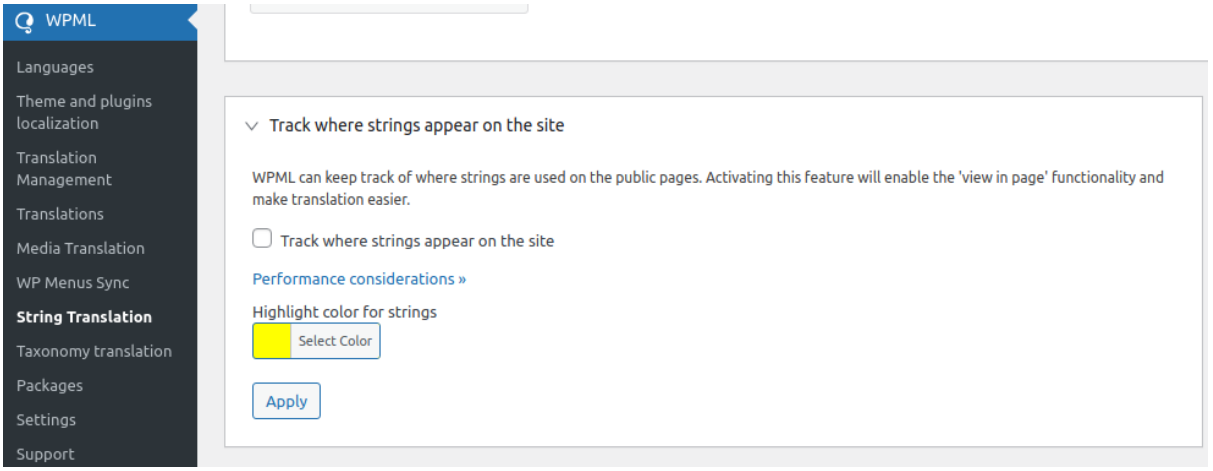
2. Remove useless scripts

```
wp_deregister_style( 'wpml-tm-admin-bar' );

wp_dequeue_style( 'wpml-tm-admin-bar' );

wp_deregister_script('wpml-xdomain-data');

wp_dequeue_script('wpml-xdomain-data');

wp_dequeue_style( 'wpml-tm-admin-bar' );
```

```
/**

* Disable WPML admin bar styles on FE

* Speed optimization

*/

function {theme_prefix}_disable_wpml_styles() {

  wp_deregister_script('wpml-xdomain-data');

}

add_action( 'wp_print_styles', '{theme_prefix}_disable_wpml_styles' );
```

3. Don’t track new strings

Make sure that this option is unchecked:



4. ACFML issue

ACFML plugin has some performance issues when ACF field groups are defined in local JSON files. ACFML generates unnecessary SQL “UPDATE” queries which can slow down the website.

To fix this issue add this constant in the WP config file:

```
define( 'ACFML_SCAN_LOCAL_FIELDS', false )
```

It’s important especially in a production environment.

Woocommerce

1. Memory limit - 256 MB

WordPress allocates 32 MB memory of PHP by default. It's not enough for a WooCommerce website. Make sure that it's increased to 256 MB.

2. Remove useless scripts

```
wp_dequeue_style( 'wc-block-style' );

wp_dequeue_style( 'storefront-gutenberg-blocks' );
```

3. Use required assets on specific pages only

```
function {theme_prefix}_woocommerce_scripts() {

  //remove generator meta tag

  remove_action( 'wp_head', array( $GLOBALS['woocommerce'], 'generator' ) );



  //first check that woo exists to prevent fatal errors

  if ( function_exists( 'is_woocommerce' ) ) {

    //dequeue scripts and styles

    if ( ! is_woocommerce() && ! is_cart() && ! is_checkout() ) {

      wp_dequeue_style( 'woocommerce_frontend_styles' );

      wp_dequeue_style( 'woocommerce_fancybox_styles' );

      wp_dequeue_style( 'woocommerce_chosen_styles' );
```

```
wp_dequeue_style( 'woocommerce_prettyPhoto_css' );
wp_dequeue_style( 'wcml-dropdown-0-css' );
wp_dequeue_style( 'woocommerce-layout-css' );
wp_dequeue_style( 'woocommerce-smallscreen-css' );
wp_dequeue_style( 'woocommerce-general-css' );
wp_dequeue_style( 'woocommerce-inline-inline-css' );
wp_dequeue_style( 'cms-navigation-style-base-css' );
wp_dequeue_style( 'cms-navigation-style-css' );
wp_dequeue_script( 'wc_price_slider' );
wp_dequeue_script( 'wc-single-product' );
wp_dequeue_script( 'wc-add-to-cart' );
wp_dequeue_script( 'wc-checkout' );
wp_dequeue_script( 'wc-add-to-cart-variation' );
wp_dequeue_script( 'wc-single-product' );
wp_dequeue_script( 'wc-cart' );
wp_dequeue_script( 'wc-chosen' );
wp_dequeue_script( 'woocommerce' );
wp_dequeue_script( 'prettyPhoto' );
wp_dequeue_script( 'prettyPhoto-init' );
wp_dequeue_script( 'jquery-blockui' );
wp_dequeue_script( 'jquery-placeholder' );
wp_dequeue_script( 'fancybox' );
wp_dequeue_script( 'jqueryui' );
```

```
    }

  }



}

add_action( 'wp_enqueue_scripts', '{theme_prefix}_woocommerce_scripts', 999 );
```

```
/**

* Disable wishlist scripts on home page

*/

function {theme_prefix}_wishlist_scripts() {

  if ( is_front_page() ) {

    wp_dequeue_script( 'jquery' );

    wp_dequeue_style( 'yith-wcwl-font-awesome' );

    wp_deregister_style( 'jquery-selectBox' );

    wp_dequeue_style( 'wcml-dropdown-0' );

    wp_dequeue_style( 'woocommerce-layout' );

    wp_dequeue_style( 'woocommerce-smallscreen' );

    wp_dequeue_style( 'woocommerce-general' );

    wp_dequeue_style( 'woocommerce-inline-inline' );

    wp_dequeue_style( 'wpml-tm-admin-bar' );

    wp_dequeue_style( 'cms-navigation-style-base' );

    wp_dequeue_style( 'cms-navigation-style' );
```

```
    wp_dequeue_style( 'cms-navigation-style' );

    wp_deregister_script( 'wcml-multi-currency' );

    wp_dequeue_script( 'wcml-multi-currency' );

    wp_deregister_script( 'wcml-front-scripts' );

    wp_dequeue_script( 'wcml-front-scripts' );

  }



}

add_action( 'wp_enqueue_scripts', '{theme_prefix}_wishlist_scripts', 999 );
```

AMP

The main goal of AMP is to improve mobile page load speed. If your site has a lot of users with mobile devices and it's possible to serve them specific mobile versions of pages, consider using AMP.

If it's not possible to prepare the whole AMP website, try to add AMP for blog posts. It can be done by one of WP plugins:

- https://wordpress.org/plugins/accelerated-mobile-pages/
- https://wordpress.org/plugins/amp/


pagepro