Wordpress

?????????? ????

???????? 404 ????????

Создайте файл `404.php` в папке темы.

??????????? ?????? `404.php`

```
<?php get_header(); ?>

<main class="container">
    <h1>404</h1>
    <p>Страница не найдена.</p>
    <a href="<?php echo home_url(); ?>">Вернуться на главную</a>
</main>

<?php get_footer(); ?>
```

4?? ????? ?? ???-?? ??????????? ? ????????

Нет. WordPress сам:

- определяет, что страница не существует
- отправляет HTTP-статус 404
- подключает файл `404.php`, если он есть

Если файла нет — используется `index.php`. и отсутствие страниц в буфере результатов поиска нужно определять там.

sitemap.xml

sitemap.xml

????? ????????

Начиная с WordPress 5.5, встроенный sitemap доступен по адресу:

`/wp-sitemap.xml`

Он формируется классом `WP_Sitemaps` и управляется через фильтры. Никакой отдельный файл в теме не создаётся — всё генерируется динамически.

Ниже — основные способы управления содержимым.

1?? ????????? ????????? sitemap

```
add_filter('wp_sitemaps_enabled', '__return_false');
```

2?? ?????? ?????????? ??? ??????? (????????, post)

```
add_filter('wp_sitemaps_post_types', function ($post_types) {
    unset($post_types['post']); // убираем записи
    return $post_types;
});
```

Пример для кастомного типа:

```
unset($post_types['product']);
```

3?? ?????? ?????????? (?????????, ???? ? ?.?.)

```
add_filter('wp_sitemaps_taxonomies', function ($taxonomies) {
    unset($taxonomies['post_tag']); // убрать теги
    return $taxonomies;
});
```

Убрать категории:

```
unset($taxonomies['category']);
```

4?? ?????? ????????????? ?? sitemap

```
add_filter('wp_sitemaps_add_provider', function ($provider, $name) {
    if ($name === 'users') {
        return false;
    }
    return $provider;
}, 10, 2);
```

5?? ???????? ?????? ?????? (?????????? ?????????? URL)

Можно отфильтровать сами записи:

```
add_filter('wp_sitemaps_posts_query_args', function ($args, $post_type) {

    if ($post_type === 'post') {
        $args['meta_query'] = array(
            array(
                'key'     => 'exclude_from_sitemap',
                'compare' => 'NOT EXISTS'
            )
        );
    }

    return $args;
}, 10, 2);
```

6?? ???????? ?????? ?????????? ?????? ? sitemap

```
add_filter('wp_sitemaps_posts_entry', function ($entry, $post, $post_type) {

    $entry['priority'] = 0.8;
    $entry['changefreq'] = 'weekly';

    return $entry;
}, 10, 3);
```

7?? ???????? ??????????? sitemap

Можно зарегистрировать свой провайдер:

```
add_action('init', function () {

    $provider = new WP_Sitemaps_Provider(
        'custom',
        'custom'
    );

    wp_register_sitemap_provider('custom', $provider);
```

```
});
```

(для продакшена обычно создают отдельный класс-провайдер)

? ??? ????????? ????

- В `functions.php` темы
- Или лучше — в собственном мини-плагине

?? ?????

Если установлен SEO-плагин вроде:

- Yoast SEO
- Rank Math
- All in One SEO

то встроенный sitemap WordPress обычно отключается, и управление происходит через плагин.

?????? ? ????????????? sitemap

В контексте WordPress **провайдер sitemap** — это класс, который отвечает за генерацию определённого набора URL внутри XML-карты сайта.

Проще говоря:

Провайдер = источник данных для sitemap.

? ??? ??? ???????? ?????? WordPress

Начиная с версии 5.5 в WordPress появился класс:

`WP_Sitemaps`

Он управляет всей системой sitemap и подключает **провайдеры**.

Каждый провайдер отвечает за свой тип данных.

По умолчанию есть три:

Провайдер	Что генерирует
`posts`	записи, страницы, custom post types
`taxonomies`	категории, теги, кастомные таксономии
`users`	архивы авторов

? ?????? ?????????

`/wp-sitemap.xml`
Это индекс. В нём ссылки на:

`/wp-sitemap-posts-post-1.xml`
`/wp-sitemap-taxonomies-category-1.xml`
`/wp-sitemap-users-1.xml`

Каждый из этих файлов формируется **отдельным провайдером**.

? ??? ?????? ????????? ??????????

Каждый провайдер — это класс, наследующий:

`WP_Sitemaps_Provider`

Он обязан реализовать два метода:

```
get_url_list( $page_num, $subtype )
get_max_num_pages( $subtype )
```

? get_url_list()

Возвращает массив URL для текущей страницы sitemap.

? get_max_num_pages()

Возвращает количество страниц sitemap (если URL много).

? ??????? ????????

Представьте sitemap как каталог:

- Индекс — оглавление
- Провайдер — глава каталога
- get_url_list() — список страниц в главе

? ?????? ??? ??????

Провайдеры позволяют:

- отключить стандартные данные
- добавить собственные URL
- изменить логику выборки
- добавить кастомный sitemap (например, для API или фильтров)

? ????????? ??????????? ??????????

Добавьте в `functions.php` или в плагин:

```
add_filter('wp_sitemaps_add_provider', function ($provider, $name) {
  // отключаем стандартные: posts, taxonomies, users
```

```
  if (in_array($name, ['posts', 'taxonomies', 'users'])) {
    return false;
  }
  return $provider;
}, 10, 2);
```

Теперь `/wp-sitemap.xml` станет пустым

? ??????? ???? ?????????

Создадим собственный класс.

```
class My_Custom_Sitemap_Provider extends WP_Sitemaps_Provider {

    public function __construct() {
        $this->name = 'custom';
        $this->object_type = 'custom';
    }

    /**
     * Количество страниц sitemap
     */
    public function get_max_num_pages( $subtype = '' ) {
        return 1;
    }

    /**
     * Список URL для sitemap
     */
    public function get_url_list( $page_num, $subtype = '' ) {

        $urls = [];

        $args = [
            'post_type'      => 'post',
            'post_status'    => 'publish',
            'posts_per_page' => -1
        ];

        $posts = get_posts($args);
```

```
        foreach ($posts as $post) {
            $urls[] = [
                'loc'     => get_permalink($post),
                'lastmod' => get_the_modified_date('c', $post),
            ];
        }

        return $urls;
    }
}
```

? ???????????? ?????????

```
add_action('init', function () {

    wp_register_sitemap_provider(
        'custom',
        new My_Custom_Sitemap_Provider()
    );

});
```

? ??? ?????????

Теперь будет доступен:

`/wp-sitemap-custom-1.xml`

И он будет содержать только ваши данные.

????????? ??????? ?????????? ?????????? ??????????

1?? ????????? ????????? sitemap

`add_filter('wp_sitemaps_enabled', '__return_false');`

2?? ?????? ?????????? ??? ??????? (????????, post)

```
add_filter('wp_sitemaps_post_types', function ($post_types) {
  unset($post_types['post']); // убираем записи return $post_types;
```

```
});
```

Пример для кастомного типа:

`unset($post_types['product']);`

3?? ?????? ?????????? (?????????, ???? ? ?.?.)

```
dd_filter('wp_sitemaps_taxonomies', function ($taxonomies) {
  unset($taxonomies['post_tag']); // убрать теги return $taxonomies;
});
```

Убрать категории:

`unset($taxonomies['category']);`

4?? ?????? ????????????? ?? sitemap

```
add_filter('wp_sitemaps_add_provider', function ($provider, $name) {
  if ($name === 'users') {
    return false;
  } return $provider;
}, 10, 2);
```

5?? ???????? ?????? ?????? (?????????? ?????????? URL)

Можно отфильтровать сами записи:

```
add_filter('wp_sitemaps_posts_query_args', function ($args, $post_type) {
   if ($post_type === 'post') {
     $args['meta_query'] = array( array( 'key' => 'exclude_from_sitemap', 'compare' => 'NOT
EXISTS' ) );
   }
   return $args;
}, 10, 2);
```

6?? ???????? ?????? ?????????? ?????? ? sitemap

```
add_filter('wp_sitemaps_posts_entry', function ($entry, $post, $post_type) {
   $entry['priority'] = 0.8;
   $entry['changefreq'] = 'weekly';
   return $entry;
}, 10, 3);
```

???????? ??????????? sitemap.xml

В статье рассматривается следующий сценарий:

- Сайт имеет только одну страницу - главную, остальные страницы это посты.
- На сайте установлен самописный плагин, который обрабатывает содержимое страницы или поста и можно получить либо русскую версию текста, либо английскую
- Русская версия текста вызывается обычной ссылкой на страницу или пост:

https://example.com/post-slug

- Английская версия вызывается добавлением к обычной ссылке окончания "/en":

https://example.com/post-slug/en

- Главная страница обязательно имеет английскую версию
- Для постов имеется поле ACF - "is_english", которое определяет, есть ли у страницы английская версия, если английской версии нет, то при ее запросе отображается 404 страница
- Для сайта формируется кастомный файл sitemap, который будет показан поисковикам через robots.txt или явным указанием в настройках индексирования.
- **На сайте не используются плагины SEO**

?????????? ???????????? wp-sitemap.xml

```
add_filter('wp_sitemaps_enabled', '__return_false');
```

Регистрация кастомного sitemap.xml

???? wordpress

Ядро wordpress

??? ????????? is_category() ? is_tag()

Это **conditional tags**, которые проверяют состояние **главного запроса** (`WP_Query`) после того, как WordPress распарсил URL.

Они становятся `true`, если текущий запрос определён как:

- архив категории → `is_category() === true`
- архив метки → `is_tag() === true`

? ??? ?????????? ??? ??????? ??????

Например URL:

```
/portfolio/category/contract/
/portfolio/topics/asp/
```

После rewrite WordPress превращает это во внутренний запрос:

```
index.php?category_name=contract
```

или

```
index.php?post_tag=asp
```

Далее `WP_Query` анализирует параметры и выставляет флаги:

```
$query->is_category = true;
$query->is_archive  = true;
```

или

```
$query->is_tag = true;
$query->is_archive = true;
```

И только после этого начинают работать:

```
is_category()
is_tag()
is_archive()
```

? ?????: ??? ??? ???????? ?????????

Conditional tags корректно работают:

- после того как WordPress разобрал запрос
- внутри `pre_get_posts`
- внутри `template_redirect`
- внутри шаблонов

 Они НЕ работают корректно внутри `init`, потому что запрос ещё не разобран.

? ?????? ????? ????????? ????? `$query`

В `pre_get_posts` безопаснее писать так:

```
add_action('pre_get_posts', function ($query) {
    if (is_admin() || !$query->is_main_query()) {
        return;
    }

    if (
        get_query_var('lang') === 'en' &&
        ( $query->is_category() || $query->is_tag() )
    ) {
        ...
    }

});
```

Почему?

Потому что:

- глобальная `is_category()` работает через `$wp_query`
- внутри `pre_get_posts` мы должны работать с конкретным `$query`

Это более правильно архитектурно.

? ??? ?????? ????????? `is_category()`

По сути это проверка:

`return $wp_query->is_category;`

Если хотите проверить руками:

```
global $wp_query;
var_dump($wp_query->is_category);
```

? ????

`is_category()` и `is_tag()`:

- зависят от разобранного запроса
- работают после `parse_query`
- безопасны в `pre_get_posts` и позже
- не зависят от вашего `lang`, если rewrite правильный

Ядро wordpress

????????? $wp_query->set_404()

После выполнения:

```
$wp_query->set_404();
```

WordPress:

- устанавливает `$wp_query->is_404 = true`
- сбрасывает другие флаги (is_single, is_page, is_archive и т.д.)
- меняет внутреннее состояние запроса

? ??? ?????????, ??? ?????? ???? 404

? 1. ????? ???????? ??????? (?????????????)

```
if ( is_404() ) {
    // это 404
}
```

Это самый правильный способ.

? 2. ????? ?????? `$wp_query`

```
global $wp_query;

if ( $wp_query->is_404 ) {
    // это 404
}
```

или

```
if ( $wp_query->is_404() ) {
    // тоже работает
}
```

? ?????

После `set_404()`:

```
$wp_query->is_single     → false
$wp_query->is_page       → false
$wp_query->is_archive    → false
$wp_query->is_category   → false
$wp_query->is_tag        → false
$wp_query->is_404        → true
```

То есть WP "переквалифицирует" запрос полностью.

? ?????? ??????

Если вы проверяете это внутри `pre_get_posts`, то `is_404()` работать не будет корректно, потому что:

- `pre_get_posts` выполняется ДО финального определения типа запроса
- 404 логика применяется позже

Проверять 404 корректно в:

```
template_redirect
wp
template_include
```

? ??????

```
add_action('template_redirect', function() {
    if ( is_404() ) {
        error_log('Это 404 страница');
    }
});
```

? ???? ????? ??????: ?? ???? ?????????? 404 ??? WordPress

Можно добавить свой флаг:

```
global $wp_query;


$wp_query->set_404();
$wp_query->custom_en_404 = true;
```

И потом:

```
if ( !empty($wp_query->custom_en_404) ) {
    // это именно ваша логика
}
```

Ядро wordpress

????? ?? ?????? link rel="shortlink" ?? ????? <head> ?????????

Да, `shortlink` **можно и лучше убрать**, для SEO он **не нужен** и иногда мешает диагностике.

? ??? ????? `rel="shortlink"`

```
<link rel="shortlink" href="https://voronkov.org/?p=108" />
```

Это:

- техническая «короткая ссылка» WordPress
- нужна была для старых мобильных клиентов и API
- **не используется поисковиками для SEO**

Google и Яндекс:

- **игнорируют** `shortlink`
- не считают его каноническим

? ?????? ??? ????? ??????

1. **Засоряет** `<head>`
2. Может сбивать инструменты SEO-анализа
3. Указывает на **некрасивый URL с** `?p=`
4. Может использоваться как **альтернативный URL** в отчётах
5. **Пользы — 0**

? ????????? ?? ???????

✔ Да
✔ WordPress продолжит работать
✔ API не ломается
✔ SEO — только лучше

? ??? ?????? `shortlink` ? WordPress

Добавь в `functions.php` темы или в свой плагин:

```
remove_action('wp_head', 'wp_shortlink_wp_head', 10);
remove_action('template_redirect', 'wp_shortlink_header', 11);
```

После этого:

```
<link rel="canonical" href="https://voronkov.org/production-planning/" />
```

— и больше ничего лишнего.

?? ?????? ??????

Если используются SEO-плагины (Yoast / RankMath):

- они **могут возвращать shortlink**
- тогда отключать нужно в настройках плагина

? ????

✔ `rel="shortlink"` не нужен
✔ Можно удалять без последствий
✔ Упростит SEO-картину
✔ Каноникал остаётся главным

Ядро wordpress

?????????? ????????? ????????? ????????? ?? ?????? ????? ??? wordpress ? ????? ? ? ?????? ? React

Ниже — **полная архитектура переводов в WordPress**:

1️⃣ как это работает **в целом**
2️⃣ как это работает **в PHP**
3️⃣ как это работает **в React / Gutenberg**
4️⃣ как соединяются **PHP + JS переводы**
5️⃣ как это обычно делают профессиональные плагины.

---

1. ????? ?????????? ????????? ? WordPress

Система переводов в WordPress состоит из 4 уровней:

```
код → POT → PO → MO / JSON
```

1?? ???

В коде используются функции:

PHP:

```
__()
_e()
_n()
_x()
```

JS:

```
wp.i18n.__()
wp.i18n._n()
wp.i18n._x()
```

2?? POT (template)

.pot — шаблон переводов.

Пример:

```
nv-seo.pot
```

Он содержит **все строки из кода**.

3?? PO

Файл конкретного языка:

```
ru_RU.po
de_DE.po
fr_FR.po
```

Пример:

```
msgid "Save settings"
msgstr "Сохранить настройки"
```

4?? MO / JSON

WordPress использует:

тип	для
MO	PHP
JSON	JavaScript

---

2. ??????????? ????????? ??? PHP

????????? ???????

```
plugin
 ├─ languages
 │   ├─ nv-seo.pot
 │   ├─ nv-seo-ru_RU.po
 │   ├─ nv-seo-ru_RU.mo
 │
 ├─ nv-seo.php
```

??????????? textdomain

```
add_action('plugins_loaded', function () {

    load_plugin_textdomain(
        'nv-seo',
        false,
        dirname(plugin_basename(__FILE__)) . '/languages'
    );

});
```

????????????

```
__('Settings saved', 'nv-seo')
```

или

```
echo __('Save', 'nv-seo');
```

---

3. ??????????? ????????? ??? React / JS

В React используется пакет:

```
@wordpress/i18n
```

(глобально это `wp.i18n`)

????????????

```
import { __ } from '@wordpress/i18n';

__('Save settings', 'nv-seo')
```

?????? React

```
import { __ } from '@wordpress/i18n';
import { Button } from '@wordpress/components';

export default function SaveButton() {

    return (
        <Button variant="primary">
            {__('Save settings', 'nv-seo')}
        </Button>
    );

}
```

4. ??? WordPress ????????? JS

WordPress **не использует MO для JS**.

Он использует:

```
JSON
```

пример:

```
languages/nv-seo-ru_RU-3a4f2.json
```

5. ?????? PHP ? JS

Очень важная функция:

```
wp_set_script_translations()
```

Она говорит WordPress:

> "подключи JSON переводы для этого JS файла"

??????

```
wp_enqueue_script(
    'nv-seo-admin',
    plugins_url('build/index.js', __FILE__),
    ['wp-i18n', 'wp-element', 'wp-components'],
    '1.0',
    true
);

wp_set_script_translations(
    'nv-seo-admin',
    'nv-seo',
    plugin_dir_path(__FILE__) . 'languages'
);
```

---

6. ??? ??????????? JSON

WordPress CLI генерирует его из PO.

```
wp i18n make-json languages
```

Результат:

```
nv-seo-ru_RU-xxxx.json
```

---

7. ?????? ????????? ???????????? ???????

```
nv-seo
|
├─ src
|   └─ index.js
|
├─ build
```

```
│   └─ index.js
│
├─ languages
│   ├─ nv-seo.pot
│   ├─ nv-seo-ru_RU.po
│   ├─ nv-seo-ru_RU.mo
│   ├─ nv-seo-ru_RU-xxxx.json
│
├─ nv-seo.php
└─ package.json
```

---

8. ????? ???????????

1?? ????? ???

PHP:

```
__('Save settings', 'nv-seo')
```

JS:

```
__('Save settings', 'nv-seo')
```

2?? ?????????? POT

```
wp i18n make-pot . languages/nv-seo.pot
```

3?? ??????? ????????

```
ru_RU.po
```

через:

- Poedit
- Loco Translate

4?? ???????????

PO → MO

5?? ?????????? JSON

```
wp i18n make-json languages
```

9. ??? ?????? ??????? ???????

Например:

- Yoast SEO
- Rank Math
- WooCommerce

используют:

```
languages/
build/
src/
```

и **WordPress i18n CLI**.

10. ?????? ????????

?????? ?????????? textdomain

```
__('Save', 'nv-seo')
```

?? ??????????? ???????????? ??????

 плохо

```
__(text, 'nv-seo')
```

??????????? sprintf

```
sprintf(
    __('Found %d errors', 'nv-seo'),
    count
)
```

?????????? ????????

```
_x('Post', 'noun', 'nv-seo')
```

11. ????? ?????? ???????????? "Code splitting"

Если ваш интерфейс React большой, лучше:

```
code splitting
```

тогда JSON будет загружаться **по частям**.

Code splitting — это разбиение большого JavaScript-бандла на несколько файлов, которые загружаются **только тогда, когда они реально нужны**.
В WordPress это особенно полезно для **React-интерфейсов в админке**, чтобы не грузить весь JS сразу.

В сборке через **@wordpress/scripts** code splitting уже поддерживается, потому что внутри используется **Webpack**.

1. ???????? ????

Без code splitting:

```
index.js
   ↓
build/index.js (400 KB)
```

С code splitting:

```
index.js
   ↓
build/index.js
build/settings.js
build/schema.js
build/faq.js
```

Подгружается только нужный модуль.

2. ??? ??? ???????? ? React

Используется **dynamic import()**.

??????? ??????

```
import SettingsPage from './settings';
```

всё попадает в один файл.

---

dynamic import

```
const SettingsPage = import('./settings');
```

Webpack создаст отдельный chunk.

Но в React правильнее использовать **React.lazy**.

---

3. ?????? code splitting

?????????

```
src
 ├─ index.js
 ├─ pages
 │   ├─ dashboard.js
 │   ├─ schema.js
 │   └─ settings.js
```

---

index.js

```
import { lazy, Suspense } from '@wordpress/element';
import { Spinner } from '@wordpress/components';

const Dashboard = lazy(() => import('./pages/dashboard'));
const Settings = lazy(() => import('./pages/settings'));

export default function App() {

    return (
        <Suspense fallback={<Spinner />}>
            <Dashboard />
        </Suspense>
```

```
    );

}
```

4. ??? ??????? ??????

После

```
npm run build
```

в папке **build** будет:

```
build
 ├─ index.js
 ├─ dashboard.js
 ├─ settings.js
```

5. ??? WordPress ????????? chunks

Важный момент.

WordPress должен знать путь:

```
wp_enqueue_script(
    'nv-seo-admin',
    plugins_url('build/index.js', __FILE__),
    ['wp-element','wp-components','wp-i18n'],
    '1.0',
    true
);
```

Webpack автоматически будет подгружать:

```
build/dashboard.js
build/settings.js
```

через

```
import()
```

6. ??? ??? ???????? ? ????????

Без splitting:

```
index.js 400KB
```

Со splitting:

```
index.js 80KB
settings.js 100KB
schema.js 120KB
faq.js 100KB
```

---

7. ??? ??? ???????? ???????

Например в SEO-плагине можно разделить:

```
src
 ├─ index.js
 ├─ pages
 │   ├─ dashboard.js
 │   ├─ schema.js
 │   ├─ sitemap.js
 │   └─ redirects.js
```

Пользователь открывает **Schema** → грузится только:

```
schema.js
```

---

8. ?????? ? ?????????

Если используется **React Router:

```
const Settings = lazy(() => import('./pages/settings'));
const Schema = lazy(() => import('./pages/schema'));
```

---

9. ????? ???????? ???????????

Можно разбить даже **библиотеки**.

Например:

```
schema-builder.js
```

грузить только при редактировании Schema.

---

10. ?????? ?????? ??? WordPress

Чтобы chunks работали, **build папка должна содержать все файлы**:

```
build
 ├─ index.js
 ├─ index.asset.php
 ├─ 234.js
 ├─ 567.js
```

WordPress автоматически подхватывает зависимости из:

```
index.asset.php
```

---

11. ??????????? ????? asset

Правильный способ:

```
$asset = include plugin_dir_path(__FILE__) . 'build/index.asset.php';

wp_enqueue_script(
    'nv-seo-admin',
    plugins_url('build/index.js', __FILE__),
    $asset['dependencies'],
    $asset['version'],
    true
);
```

---

12. ???? ?? "Code splitting"

Code splitting позволяет:

✔ ускорить загрузку
✔ уменьшить первый бандл
✔ грузить код **по требованию**

Это стандартная практика для:

- Yoast SEO
- WooCommerce
- Rank Math

---

12. ???????? ???????? ????????

Они думают:

> JSON перевод нужно писать вручную.

Нет.

Он **генерируется автоматически**.

---

13. ????? ?????????? ???????????? ??? ?????? SEO-???????

```
nv-seo
 ├─ src
 │   └─ admin
 │       └─ index.js
 │
 ├─ build
 │   └─ admin.js
 │
 ├─ languages
 │   ├─ nv-seo.pot
 │   ├─ nv-seo-ru_RU.po
 │   ├─ nv-seo-ru_RU.mo
 │   ├─ nv-seo-ru_RU.json
 │
 └─ nv-seo.php
```

Ядро wordpress

??? ????? TRANSIENT?

В WordPress **transient** — это механизм **временного кэширования данных с автоматическим сроком жизни (TTL)**.

Проще:

```
transient = временное значение в базе (или кэше), которое само “протухает”
```

? ????? ????? transient

Чтобы **не выполнять тяжёлые операции каждый раз**:

- сложные `WP_Query`
- API-запросы
- генерация sitemap
- вычисления SEO-данных

? ??? ??? ????????

?????????

```
set_transient('my_key', $data, 3600);
```

□□ сохранить данные на **1 час**

?????????

```
$data = get_transient('my_key');
```

????????

```
delete_transient('my_key');
```

? ??? ??? ????????

```
$data = get_transient('my_key');

if ($data === false) {
    // кэша нет или истёк
    $data = expensive_function();

    set_transient('my_key', $data, 3600);
}

return $data;
```

? ??? ???????? transient

По умолчанию:

□□ в таблице:

```
wp_options
```

ключи выглядят так:

```
_transient_my_key
_transient_timeout_my_key
```

???? ???? object cache (Redis/Memcached)

Тогда transient хранятся:

□□ в памяти (быстрее)

? ????? ?????

```
set_transient('key', $data, 3600);
```

значение	смысл
60	1 минута

значение	смысл
3600	1 час
86400	1 день

? ?????

TTL — это **максимальное время**, а не гарантия:

```
transient может исчезнуть раньше
```

(например при очистке кэша)

? ??????? ?????????????

1. ??? WP_Query

```
$posts = get_transient('latest_posts');

if ($posts === false) {
    $posts = get_posts([...]);
    set_transient('latest_posts', $posts, 600);
}
```

---

2. ??? API

```
$response = get_transient('api_data');

if ($response === false) {
    $response = wp_remote_get('https://api.site.com');
    set_transient('api_data', $response, 3600);
}
```

---

3. Sitemap

```
$xml = get_transient('sitemap');

if ($xml === false) {
```

```
    $xml = generate_sitemap();
    set_transient('sitemap', $xml, 3600);
}
```

?? ?????? ???????????

1. ?? ??? ?????????? ??????

 нельзя хранить:

- настройки
- важные данные

 transient может исчезнуть

---

2. ????????????

WordPress сам удаляет истёкшие transient **не сразу**, а при обращении к ним.

---

3. ????? ???? ?????? ?????????

При:

- очистке кэша
- обновлении плагинов
- некоторых хостингах

? Transient vs Options

	transient	option
время жизни	есть	нет
может исчезнуть	да	нет
назначение	кэш	постоянные данные

? ????? ???????????

Используй transient, если:

✔ данные тяжело получать
✔ можно пересчитать
✔ не критично потерять

? ????? ?? ????????????

- SEO мета
- важные даты
- пользовательские настройки

? ????

Transient — это встроенный кэш WordPress с TTL

Он:

✔ ускоряет сайт
✔ снижает нагрузку
✔ автоматически “протухает”

???? wordpress, hooks

Ядро wordpress, hooks

Filter document_title_parts

`document_title_parts` — это **низкоуровневый фильтр WordPress**, который управляет **частями заголовка** `<title>`, *до* того как они будут склеены в одну строку.

Разберём подробно.

? ??? WordPress ????????? `<title>`

Современный WordPress (4.4+) делает так:

1. Собирает массив частей заголовка
2. Пропускает его через фильтр `document_title_parts`
3. Склеивает части через разделитель
4. Пропускает итог через `pre_get_document_title`
5. Выводит `<title>`

Схема:

```
document_title_parts  →  pre_get_document_title  →  <title>
```

? ??? ????? `document_title_parts`

Это фильтр, который возвращает **массив**:

```
array(
    'title'   => 'Заголовок страницы',
    'page'    => 'Page 2',
    'tagline' => 'Описание сайта',
    'site'    => 'Название сайта',
);
```

WordPress сам решает, какие части включать.

? ?????? ???????????? ???????

??? ?????:

```
array(
    'title' => 'Название поста',
    'site'  => 'Название сайта',
);
```

??? ???????:

```
array(
    'site'    => 'Название сайта',
    'tagline' => 'Описание сайта',
);
```

? ??? ??????????? `document_title_parts`

??????: ???????? ?????? title

```
add_filter('document_title_parts', function ($parts) {

    if (is_singular()) {
        $seo_title = get_field('seo_title');

        if ($seo_title) {
            $parts['title'] = $seo_title;
        }
    }

    return $parts;
});
```

✔ меняется только заголовок
✔ имя сайта остаётся
✔ слоган остаётся (если включён)

? ??????: ?????? ???????? ?????

```
add_filter('document_title_parts', function ($parts) {

    unset($parts['site']);

    return $parts;
```

```
});
```

Результат:

```
<title>Заголовок страницы</title>
```

? ??????: ????????? ???????

```
add_filter('document_title_parts', function ($parts) {

    return array(
        'title' => $parts['title'],
        'site'  => $parts['site'],
    );
});
```

? document_title_parts vs pre_get_document_title

Фильтр	Что меняет	Когда использовать
`document_title_parts`	части title	тонкая настройка
`pre_get_document_title`	весь title строкой	полный контроль

□□ SEO-плагины используют оба.

? ????? ???????

- `document_title_parts` не выводит `<title>`
- он **НЕ строка**, а массив
- `pre_get_document_title` срабатывает позже

? ???????????? ??? SEO

Для кастомного SEO:

- использовать `document_title_parts` → управлять логикой
- использовать `pre_get_document_title` → крайний приоритет

? ????

✔ `document_title_parts` — управляет частями `<title>`
✔ Работает до склейки
✔ Позволяет убрать site/tagline/page
✔ Идеален для SEO без дублирования

Ядро wordpress, hooks

Action wp_head

? ??????????? SEO Description

? ?????????? ?????? ? WordPress (????? ??????)

 НЕ делаем `echo <meta>` в шаблоне
✔ используем `wp_head`

? ????????????? ??? (?????, ????????, ??????????)

```
add_action('wp_head', function () {

    $description = '';

    // Посты и страницы
    if (is_singular()) {
        $description = get_field('seo_desc');
    }
    // Категории и метки
    elseif (is_category() || is_tag()) {
        $term = get_queried_object();
        $description = get_field('seo_desc', $term);
    }

    // Фолбэк — excerpt или описание термина
    if (!$description) {
        if (is_singular()) {
            $description = get_the_excerpt();
        }
        elseif (is_category() || is_tag()) {
            $term = get_queried_object();
            $description = term_description($term);
        }
```

```
    }

    if (!$description) {
        return;
    }

    // Чистим HTML и лишние пробелы
    $description = wp_strip_all_tags($description);
    $description = trim(preg_replace('/\s+/', ' ', $description));

    echo '<meta name="description" content="' . esc_attr($description) . '">' . "\n";

});
```

? ?????? ?????? ???

✔ Работает для:

- постов
- страниц
- категорий
- меток

✔ Без дублей
✔ Без XSS
✔ Совместимо с темами и плагинами

? ?????? ???????? SEO Description

- 120–160 символов (RU)
- 70–155 (моб)
- уникальный
- без кавычек
- **не повторять title**
- коммерческий посыл (если уместно)

? ?????? ??????

 несколько `<meta description>`
 HTML внутри

 слишком длинный текст
 одинаковый на всех страницах

? ???? ????? ????????

Можно добавить:

- `seo_desc_en` для EN версии
- авто-обрезку до 160 символов
- приоритет для главной
- проверку `noindex`
- фильтр `wpseo_metadesc` (если используешь Yoast)

? ????

✔ Meta description добавляется через `wp_head`
✔ Используется ACF
✔ Есть fallback
✔ Без конфликтов

???????

Плагины

Advanced Custom Fields (ACF)

Рассматривается обычная версия плагина, не PRO.

????.

acf/init

Через него можно добавлять поля к различынм сущностям wordpress.

?????????? ???? ????? ? ?????????? ??????:

```
add_action('acf/init', function() {

    acf_add_local_field_group(array(
        'key' => 'group_category_fields',
        'title' => 'Category Fields',
        'fields' => array(
            array(
                'key' => 'seo_title_category',
                'label' => 'SEO Title',
                'name' => 'seo_title',
                'type' => 'text',
            ),
            array(
                'key' => 'seo_desc_category',
                'label' => 'SEO Description',
                'name' => 'seo_desc',
                'type' => 'textarea',
            ),
        ),
        'location' => array(
            array(
                array(
                    'param' => 'taxonomy',
```

```
                    'operator' => '==',
                    'value' => 'category',
                ),
            ),
        ),
    ));

});
```

В этом примере добавляется текстовое поле и текстовый блок.

Admin-?????? wordpress

????? frameworks ????????????? ? ?????????? ?????? wordpress, ????? ?? ????? ???? ????????????? ??? ?????????? ??????????? admin-???????

> В «коробочной» версии WordPress **нет полноценного JS-фреймворка как Laravel + Vue из коробки**,
> но есть встроенные инструменты, которые можно использовать.

? 1?? Gutenberg (React) — ??????? ?????????? framework

Внутри админки WordPress используется **React**.

WP поставляется с:

- React
- ReactDOM
- wp.element
- wp.components
- wp.data
- wp.apiFetch
- wp.i18n
- wp.hooks

Это называется **WordPress JavaScript Packages**.

?????? ???????????:

```
wp_enqueue_script(
    'my-admin-script',
    plugins_url('admin.js', __FILE__),
    ['wp-element', 'wp-components', 'wp-api-fetch'],
    '1.0',
    true
);
```

И в JS:

```
const { Button, TextControl } = wp.components;
const { useState } = wp.element;
```

Это официальный способ.

✔ современный
✔ нативный
✔ безопасный
✔ используется в Gutenberg

Если делать сложный интерфейс — это лучший путь.

? 2?? jQuery (??????? ?? ?????????)

WordPress до сих пор включает jQuery.

```
wp_enqueue_script('jquery');
```

Подходит для:

- простых форм
- динамических элементов
- AJAX

Но:

 это уже устаревший подход
 не подходит для сложного SPA-интерфейса

? 3?? jQuery UI (???????? ???????)

В комплекте есть:

- Tabs
- Accordion
- Dialog
- Datepicker

Подключается так:

```
wp_enqueue_script('jquery-ui-tabs');
```

Но выглядит немного устаревшим визуально.

? ??? ?????????? "???"

Технически можно перечислить всё вручную:

add_action('admin_enqueue_scripts', function () {

wp_enqueue_script('jquery');
wp_enqueue_script('jquery-ui-core');
wp_enqueue_script('jquery-ui-widget');
wp_enqueue_script('jquery-ui-mouse');
wp_enqueue_script('jquery-ui-position');

wp_enqueue_script('jquery-ui-draggable');
wp_enqueue_script('jquery-ui-droppable');
wp_enqueue_script('jquery-ui-resizable');
wp_enqueue_script('jquery-ui-selectable');
wp_enqueue_script('jquery-ui-sortable');

wp_enqueue_script('jquery-ui-accordion');
wp_enqueue_script('jquery-ui-autocomplete');
wp_enqueue_script('jquery-ui-button');
wp_enqueue_script('jquery-ui-datepicker');
wp_enqueue_script('jquery-ui-dialog');
wp_enqueue_script('jquery-ui-menu');
wp_enqueue_script('jquery-ui-progressbar');
wp_enqueue_script('jquery-ui-slider');
wp_enqueue_script('jquery-ui-spinner');
wp_enqueue_script('jquery-ui-tabs');
wp_enqueue_script('jquery-ui-tooltip');

});

⚠ Но это плохая практика.

? ?????? ?? ????? ?????????? ???

1. Лишняя нагрузка
2. Замедление админки
3. Возможные конфликты
4. Большинство модулей тебе не понадобится

? ?????????? ??????

Подключать только то, что нужно.

Например:

Tabs:

wp_enqueue_script('jquery-ui-tabs');

Dialog:

wp_enqueue_script('jquery-ui-dialog');
wp_enqueue_style('wp-jquery-ui-dialog');

Datepicker:

wp_enqueue_script('jquery-ui-datepicker');

? ?????: CSS

WordPress **не включает полный jQuery UI theme CSS**.

Для dialog есть:

wp_enqueue_style('wp-jquery-ui-dialog');

Но для tabs, accordion и других — стили придётся писать самому или подключать свою тему jQuery UI.

? 4?? Dashicons

Встроенная иконная библиотека.

```
wp_enqueue_style('dashicons');
```

? 5?? ??????????? ????? ???????

Можно использовать встроенные CSS-классы:

- `.wrap`
- `.form-table`
- `.button`
- `.notice`
- `.widefat`
- `.nav-tab-wrapper`
- `.postbox`

Пример:

```
<div class="wrap">
    <h1>SEO Settings</h1>
</div>
```

Это самый простой путь.

? 6?? @wordpress/components (UI-??????????)

Очень мощная вещь.

Компоненты:

- Button
- TextControl
- SelectControl
- ToggleControl
- Panel
- TabPanel
- Notice
- Modal

Пример:

```
wp.element.render(
    wp.element.createElement(
        wp.components.Button,
        { isPrimary: true },
        'Save'
    ),
    document.getElementById('app')
);
```

Это нативный WP-UI на React.

? ???? ??? ? ???????

- Vue
- Angular
- Bootstrap
- Tailwind
- Material UI

Их можно подключить вручную, но это уже не «из коробки».

????????:

? ??????? ?????????

→ стандартные WP формы + PHP
(самый быстрый способ)

? ??????? ???????

→ jQuery + AJAX

? ???????????????

→ React + wp.components

? ????????????? API WordPress

Ты можешь использовать:

- REST API (`register_rest_route`)
- Settings API
- Options API
- AJAX (`admin-ajax.php`)

- Nonce security