# ???? wordpress

- Как работают is_category() и is_tag()
- Установка $wp_query->set_404()
- Можно ли убрать link rel="shortlink" из блока <head> страницы?
- Архитектура поддержки переводов интерфейсов на другие языки для wordpress в целом и в связке с React
- Что такое TRANSIENT?

# ??? ???????? 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

# ????????? $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
}
```

# ????? ?? ?????? 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

# ?????????? ????????? ????????? ????????? ?? ?????? ????? ??? 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
```

# ??? ????? 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 |
| 3600 | 1 час |
| 86400 | 1 день |

# ? ?????

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

```
transient может исчезнуть раньше
```

(например при очистке кэша)

---

# ? ??????? ?????????????

## 1. ??? WP_Query

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

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

## 2. ??? API

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

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

## 3. Sitemap

```php
$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”