Authentication and Authorization

- Understanding GraphQL Operations
- Authentication & Authorization Concepts
- Authentication Methods
  - 1. Remote HTTP Requests
  - 2. Cookie-Based Authentication (Browser Requests)
    - Why Nonces Are Required
    - How to Include the Nonce
    - How to Get a Nonce
    - Requests Without a Valid Nonce
    - Disabling Nonce Requirement (Development Only)
    - Customizing the Error Status Code
    - When to Use Cookie Auth vs. Other Methods
  - 3. Direct PHP Function Calls
- Authorization in WPGraphQL
  - Field-Level Authorization
  - Mutation Authorization

Understanding GraphQL Operations

Before diving into authentication and authorization, it's important to understand how GraphQL operations work in WPGraphQL.

GraphQL has two main operation types:

- **Queries**: Used for fetching data
- **Mutations**: Used for modifying data

While mutations use the `mutation` keyword and must be processed synchronously (one after another), both queries and mutations are fundamentally similar - they map input to resolver functions that interact with WordPress.

Authentication & Authorization Concepts

- **Authentication**: The process of verifying a user's identity (logging in, validating credentials)
- **Authorization**: The process of verifying what a user can access or modify (permissions)

In WPGraphQL, these processes build on WordPress's existing user and capability systems.

Authentication Methods

WPGraphQL supports multiple authentication approaches depending on your use case:

1. Remote HTTP Requests

For applications making requests to the `/graphql` endpoint, you can use:

- **Application Passwords** (Recommended for WordPress 5.6+)
  - Built into WordPress core
  - Secure token-based authentication
  - Integration Guide
- **JWT Authentication**
  - Uses JSON Web Tokens
  - Ideal for headless WordPress applications
  - WPGraphQL JWT Authentication Plugin
- **Basic Authentication**
  - Simple username/password authentication
  - Basic Auth Plugin
  - Note: Only use with SSL/HTTPS connections
- **OAuth1**

- More complex but very secure
- OAuth1 Plugin

2. Cookie-Based Authentication (Browser Requests)

When making GraphQL requests from a browser where the user is already logged into WordPress, you can use cookie-based authentication. This is common for:

- Custom admin pages
- Frontend JavaScript applications served from WordPress
- The GraphiQL IDE

Important: Cookie-authenticated requests require a **nonce** for CSRF (Cross-Site Request Forgery) protection. This matches the behavior of the WordPress REST API.

Why Nonces Are Required

Cookies are automatically sent by the browser with every request to your WordPress site. Without nonce verification, a malicious website could trick a logged-in user's browser into making GraphQL requests on their behalf. The nonce ensures the request originated from your WordPress site.

How to Include the Nonce

You can include the nonce in one of two ways:

1. **HTTP Header** (Recommended):

```
fetch('/graphql', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'X-WP-Nonce': wpApiSettings.nonce  // or your nonce variable
  },
  body: JSON.stringify({ query: '{ viewer { name } }' })
});
```

2. **Query Parameter**:

```
/graphql?query={viewer{name}}&_wpnonce=your_nonce_here
```

How to Get a Nonce

Option 1: From WordPress Admin Pages

On any WordPress admin page, the nonce is available in `wpApiSettings.nonce`:

```
const nonce = wpApiSettings.nonce;
```

Option 2: Using wp_localize_script() in Your Plugin/Theme

Pass the nonce to your JavaScript when enqueueing scripts:

```
wp_enqueue_script('my-graphql-app', 'path/to/app.js', [], '1.0', true);
wp_localize_script('my-graphql-app', 'myAppSettings', [
    'nonce' => graphql_get_nonce(),
    'graphqlEndpoint' => home_url('/graphql'),
]);
```

Then in your JavaScript:

```
fetch(myAppSettings.graphqlEndpoint, {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json',
        'X-WP-Nonce': myAppSettings.nonce,
    },
    body: JSON.stringify({ query: '{ viewer { name } }' }),
});
```

Option 3: From the GraphiQL IDE Page

The GraphiQL IDE page includes the nonce in `wpGraphiQLSettings.nonce`.

Requests Without a Valid Nonce

WPGraphQL handles missing or invalid nonces differently:

Scenario	Behavior
No nonce provided	Request is **downgraded to guest** - executes as unauthenticated user. `viewer` returns `null`, private data not accessible.
"Falsy" nonce value (`null`, `undefined`, empty string, `false`, `0`)	Treated as "no nonce" - **downgraded to guest**.
Invalid nonce provided (real but wrong/expired)	Request **fails with HTTP 403** - returns `"Cookie nonce is invalid"` error message.

This design:

- Allows public queries to still work when no nonce is provided (e.g., a logged-in user sharing a GraphQL URL)
- Gracefully handles JavaScript serialization edge cases (e.g., `JSON.stringify({ nonce: null })` → `"null"`)
- Rejects requests with tampered or expired nonces that appear to be real authentication attempts

Disabling Nonce Requirement (Development Only)

For local development or testing, you can disable the nonce requirement using a filter:

```
// WARNING: Only use in development environments!
add_filter('graphql_cookie_auth_require_nonce', function($require_nonce) {
    if (wp_get_environment_type() === 'local') {
        return false;
    }
    return $require_nonce;
});
```

⚠ **Never disable this in production** - it would expose your site to CSRF attacks.

Customizing the Error Status Code

By default, invalid nonce errors return HTTP 403 (Forbidden). If you have legacy clients that expect HTTP 200 with a GraphQL error response, you can customize this:

```
add_filter('graphql_authentication_error_status_code', function($status_code, $error) {
    return 200; // Return 200 instead of 403
}, 10, 2);
```

When to Use Cookie Auth vs. Other Methods

Use Case	Recommended Auth Method
WordPress admin pages	Cookie + Nonce
Frontend JS on same domain	Cookie + Nonce
External/headless apps	Application Passwords or JWT
Mobile apps	Application Passwords or JWT
Server-to-server	Application Passwords
CI/CD or automated scripts	Application Passwords

3. Direct PHP Function Calls

When using WPGraphQL programmatically within WordPress:

- Uses the current user's session
- Call `graphql([ 'query' => $query ])` directly
- Inherits WordPress authentication context

Authorization in WPGraphQL

WPGraphQL implements a granular authorization system:

Field-Level Authorization

- Each field can have its own authorization rules
- Fields can return null if user lacks permission
- Other fields in the same query still resolve if authorized
- Examples:
  - `email` in `generalSettings` requires authentication
  - `{ posts( where: { status: DRAFT } ) { nodes { id, title } } }` requires authentication (draft posts are not public)

Mutation Authorization

- Checks WordPress capabilities before executing
- Respects WordPress roles and permissions
- Examples:
  - Creating a post checks `publish_posts` capability

□□□ #1
□ □□□ □□□ 2026-08-29 23:36:26 UTC
□ □□□ □□□ 2026-08-29 23:36:26 UTC