Global Blocks Implementation Report

Summary

Successfully implemented a Global Blocks system for reusable content across pages in a headless WordPress + Next.js setup.

What Went Wrong Initially

1. Duplicate Field Groups

- **Problem**: Created both PHP-based and JSON-imported ACF field groups
- **Symptom**: Conflicting field definitions, GraphQL schema issues
- **Solution**: Use single source - either PHP or JSON, not both

2. Multiple Options Pages

- **Problem**: Created separate "Global Blocks" page when "Global Content" already existed
- **Symptom**: Fields attached to wrong options page
- **Solution**: Use existing options page structure

3. Incorrect GraphQL Queries

- **Problem**: Used `acfOptionsGlobalContent` instead of `globalOptions`
- **Symptom**: "Cannot query field" errors
- **Solution**: Use correct field names from GraphQL schema

4. Image Field Structure Issues

- **Problem**: Queried `image { sourceUrl }` instead of `image { node { sourceUrl } }`
- **Symptom**: GraphQL field type errors
- **Solution**: Follow ACF GraphQL image field convention

Correct Implementation Structure

WordPress Backend

```php
```

```php
// 1. Single Options Page
acf_add_options_page(array(
    'page_title' => 'Global Content',
    'menu_title' => 'Global Content',
    'menu_slug' => 'global-content',
    'show_in_graphql' => true,
    'graphql_field_name' => 'globalOptions'
));

// 2. Field Group Structure
acf_add_local_field_group(array(
    'key' => 'group_global_shared_content',
    'title' => 'Global Shared Content',
    'fields' => array(
        array(
            'key' => 'field_global_why_cda_block',
            'name' => 'why_cda_block',
            'type' => 'group',
            'show_in_graphql' => 1, // CRITICAL
            'sub_fields' => array(
                // title, subtitle, cards repeater
            )
        )
    ),
    'location' => array(
        array(
            array(
                'param' => 'options_page',
                'operator' => '==',
                'value' => 'global-content'
            )
        )
    ),
    'show_in_graphql' => 1,
    'graphql_field_name' => 'globalSharedContent'
));
```

GraphQL Query Structure

javascript

```
// ✅ CORRECT
export const GET_HOMEPAGE_CONTENT = gql`
  query GetHomepageContent($id: ID!) {
    page(id: $id, idType: DATABASE_ID) {
      id
      title
      homepageContent {
        // page-specific fields
      }
    }
    globalOptions {                  // Options page
      globalSharedContent {          // Field group
        whyCdaBlock {                // Block group
          title
          subtitle
          cards {                    // Repeater
            title
            description
            image {                  // Image field
              node {                 // Required wrapper
                sourceUrl
                altText
              }
            }
          }
        }
      }
    }
  }
`;
```

React Component Pattern

javascript

```
// ✅ CORRECT
const WhyCdaBlock = ({ globalData }) => {
  if (!globalData?.cards || globalData.cards.length === 0) return null;

  return (
    <section>
      <h2>{globalData.title}</h2>
      <p>{globalData.subtitle}</p>
      {globalData.cards.map((card, index) => (
        <div key={index}>
          <img src={card.image.node.sourceUrl} alt={card.image.node.altText} />
          <h3>{card.title}</h3>
          <p>{card.description}</p>
        </div>
      ))}
    </section>
  );
};

// Usage
<WhyCdaBlock
  globalData={pageData.globalOptions?.globalSharedContent?.whyCdaBlock}
/>
```

Key Requirements for Success

ACF Configuration

- All fields must have `'show_in_graphql' => 1`
- Field groups need `'show_in_graphql' => 1`
- Options page needs `'show_in_graphql' => true`
- Consistent naming between ACF and GraphQL

GraphQL Schema Verification

1. Test basic options query: `{ globalOptions { __typename } }`
2. Check field availability: `{ __type(name: "GlobalOptions") { fields { name } } }`
3. Verify complete path: `globalOptions.globalSharedContent.whyCdaBlock`

Debugging Process

1. **Schema First**: Always verify fields exist in GraphQL schema

2. **Step by Step**: Test each level of nesting separately
3. **Field Structure**: Check ACF field types match query expectations
4. **Image Fields**: Use `node { sourceUrl, altText }` pattern

Common Pitfalls to Avoid

1. **Don't** create duplicate field definitions
2. **Don't** mix PHP and JSON field registration
3. **Don't** forget `show_in_graphql` flags
4. **Don't** assume GraphQL field names match ACF names
5. **Don't** query images without `node` wrapper

File Structure

```
src/
├── components/
│   └── GlobalBlocks/
│       └── WhyCdaBlock.js     // Reusable component
├── lib/
│   └── graphql/
│       └── queries.js         // GraphQL queries
└── app/
    └── page.js                // Page implementation

wp-content/
└── plugins/
    └── cda-cms.php            // ACF field definitions
```

Benefits Achieved

- ✅ Single source of truth for global content
- ✅ Reusable across all pages
- ✅ Easy content management in WordPress
- ✅ Type-safe GraphQL queries
- ✅ Scalable component system

Next Steps for Additional Global Blocks

1. Add new field groups to existing `globalSharedContent`

2. Follow same naming convention: `global[BlockName]Block`
3. Create corresponding React components
4. Test GraphQL schema before implementation