WordPress Plugin Project Structure

PHP-based WordPress plugin with admin settings, hooks, shortcodes, and optional Gutenberg blocks.

#wordpress #plugin #php #cms #gutenberg

PNG PDF Copy Prompt

Project Directory

```
my-plugin/
  my-plugin.php Main plugin file
  includes/ Core PHP classes
    class-plugin.php Main plugin cla…
    class-loader.php Hook loader
    class-activator.php Activation logic
    class-deactivator.php Deactivation cl…
  admin/ Admin-facing co…
    class-admin.php Admin hooks and…
    partials/ Admin view temp…
      settings-page.php
    css/
      admin.css
    js/
      admin.js
  public/ Public-facing c…
    class-public.php Frontend hooks
    partials/
      shortcode-output.php
    css/
      public.css
    js/
      public.js
  blocks/ Gutenberg block…
    my-block/
      block.json Block metadata
      index.js Block registrat…
      edit.js Editor component
      save.js Frontend render
      style.css
  languages/ Translation fil…
    my-plugin.pot
  uninstall.php Cleanup on unin…
  readme.txt WordPress.org r…
  composer.json PHP dependencies
  package.json For block assets
  .gitignore
```

Why This Structure?

This structure separates admin and public code, uses a class-based architecture, and follows WordPress coding standards. The `includes/` folder contains shared logic, while `admin/` and `public/` handle their respective contexts.

Key Directories

`my-plugin.php` - Plugin header, bootstrap, and activation hooks

`includes/` - Core classes and shared functionality

`admin/` - Admin menus, settings pages, and assets

`blocks/` - Gutenberg blocks with React components

Main Plugin File

```
run();
```

Getting Started

1. Create plugin folder in `wp-content/plugins/`
2. Add main PHP file with plugin header comment
3. Create `includes/class-plugin.php` with hook registration
4. Activate plugin in WordPress admin
5. Use `wp scaffold plugin` for quick scaffolding

WordPress Hooks

`Actions` - `add_action('init', ...)` to run code at specific points

`Filters` - `add_filter('the_content', ...)` to modify data

`Shortcodes` - `add_shortcode('my_tag', ...)` for content embeds

`AJAX` - `wp_ajax_*` hooks for async requests

Best Practices

- Prefix all functions, classes, and hooks with unique slug
- Use `wp_enqueue_script/style` for assets, never inline
- Escape output with `esc_html()`, `esc_attr()`, etc.
- Use nonces for form security
- Make strings translatable with `__()` and `_e()`

Common APIs

`Options API` - `get_option()`, `update_option()` for settings

`Settings API` - Built-in settings pages with validation

`REST API` - `register_rest_route()` for custom endpoints

`Custom Post Types` - `register_post_type()` for custom content

When To Use This

- Adding features to WordPress sites
- Custom admin functionality
- Integrating third-party services
- Custom post types and taxonomies
- Gutenberg editor extensions

Trade-offs

`PHP required` - Must know PHP for core functionality

`Global namespace` - Risk of conflicts without proper prefixing

`Review process` - WordPress.org has strict review guidelines