ROOTED DEV GUIDES

WordPress Plugin Build Planner

A free planning and pre-release checklist for turning a plugin idea into a safer, cleaner, more maintainable WordPress project.

FREE COMPANION RESOURCE

Companion to Book 3 - Building WordPress Plugins From the Root Up

+ Define the problem before writing code.
+ Plan data, permissions, security, settings, and lifecycle behavior.
+ Use a practical pre-release checklist before shipping.

How to use this: Work through the pages, mark what applies to you, and keep the PDF as a practical reference. This resource is intentionally concise; the full book goes deeper into the why, how, and real-world implementation.

Learn WordPress from the root up.

1. Define the Plugin Before You Code

Good plugin architecture starts with a narrow problem statement.

Problem & audience

[ ] Write one sentence describing the problem the plugin solves.

[ ] Identify the primary user: site owner, editor, developer, store manager, or visitor.

[ ] List the minimum features required for version 1.0.

[ ] Move nice-to-have ideas into a later-version list.

[ ] Check whether WordPress core already provides part of the solution.

Boundaries

[ ] Define what the plugin will intentionally NOT do.

[ ] Identify external services or APIs the plugin depends on.

[ ] Decide what should happen if another plugin or service is unavailable.

[ ] Choose a unique plugin slug, text domain, option prefix, and PHP namespace/prefix.

[ ] Document minimum supported WordPress and PHP versions.

2. Plan the Data & Lifecycle

Decide where data lives and what happens when the plugin changes or leaves the site.

Data design

[ ] List every option, transient, custom table, post meta, user meta, or taxonomy the plugin creates.

[ ] Avoid creating a custom database table unless the data and query pattern justify it.

[ ] Identify which values need defaults and which are required.

[ ] Plan migrations for future schema or option changes.

[ ] Define what uninstall should delete and what it should preserve.

Lifecycle

[ ] Activation performs only work that truly belongs on activation.

[ ] Deactivation does not destroy user data.

[ ] Uninstall behavior is explicit and documented.

[ ] Scheduled events are registered safely and removed when appropriate.

[ ] Version upgrades can run without requiring users to reinstall the plugin.

3. WordPress Security Checklist

A secure plugin assumes every request and every user-controlled value can be wrong or malicious.

Permissions & requests

[ ] Privileged actions check an appropriate capability with current_user_can().

[ ] State-changing admin requests use a nonce and verify it.

[ ] AJAX and REST endpoints perform authorization on the server, not only in JavaScript.

[ ] Direct file access is blocked where appropriate.

[ ] Sensitive actions fail safely and return useful errors.

Input & output

[ ] Input is validated against the expected type, format, and allowed values.

[ ] Input is sanitized at the appropriate boundary before storage or use.

[ ] Output is escaped for its context: HTML, attribute, URL, JavaScript, or textarea.

[ ] Database queries use WordPress APIs or $wpdb->prepare() when interpolation is required.

[ ] Uploaded files are restricted to intended types, locations, and permissions.

4. Architecture & WordPress Integration

Prefer WordPress conventions and small components over one enormous plugin file.

Code structure

[ ] Bootstrap code is small and predictable.

[ ] Admin-only code is not loaded unnecessarily on the front end.

[ ] Hooks are registered in a way that is easy to trace and test.

[ ] Business logic is separated from presentation where practical.

[ ] Functions/classes use unique namespaces or prefixes to avoid collisions.

Integration

[ ] Strings shown to users are translation-ready.

[ ] Styles and scripts are enqueued rather than printed directly into pages.

[ ] Scripts/styles load only where needed when practical.

[ ] Settings use appropriate WordPress Settings API patterns or equivalent validation.

[ ] Plugin behavior respects multisite requirements if multisite is supported.

5. Pre-Release Gate

Before calling a plugin ready, test the places most likely to break in the real world.

Functional testing

[ ] Fresh install works with no existing plugin data.

[ ] Upgrade from the previous version works correctly.

[ ] Activation, deactivation, and reactivation are safe.

[ ] Uninstall behavior matches the documentation.

[ ] The plugin is tested with WP_DEBUG enabled.

Compatibility & quality

[ ] Test the minimum supported PHP version and a current PHP version.

[ ] Test the minimum supported WordPress version when practical and the current release.

[ ] Test with a default WordPress theme.

[ ] Check for PHP warnings, notices, JavaScript console errors, and failed network requests.

[ ] Run WordPress Coding Standards and Plugin Check where appropriate.

Release package

[ ] README/readme.txt accurately describes features, installation, and limitations.

[ ] Changelog explains meaningful user-facing changes.

[ ] No development secrets, debug files, local paths, or unnecessary build artifacts are included.

[ ] Screenshots and assets match the current interface.

[ ] Support and documentation links point to working destinations.

Fast planning test
Ready - You can explain the problem, scope, data model, permissions, lifecycle, and release tests before coding.
Needs work - Any answer that begins with "we will figure that out later" should become an explicit design decision before launch.

GO DEEPER WITH THE FULL GUIDE

Book 3 - Building WordPress Plugins From the Root Up

The full guide walks from plugin architecture and WordPress APIs through security, data handling, admin interfaces, debugging, packaging, and release practices with much more explanation and implementation detail.

rooteddevguides.com