My Private Site – Repository Review

1. Project Purpose & Architecture

Purpose: *My Private Site* is a WordPress plugin that locks down an entire site so only logged-in users can view contentgithub.com. Any visitor who isn't authenticated is redirected to the login page, effectively making the site private. The plugin provides options to fine-tune this behavior – for example, admins can optionally leave the homepage or certain parts of the site publicly accessible while protecting the rest.

Overall Architecture: The plugin integrates deeply with WordPress's hook system to enforce privacy. On initialization, it conditionally loads **front-end** or **admin** code based on context. In admin mode, it registers settings pages (using a library called CMB2) for configuring the plugin. In front-end mode, if "private site" is enabled, it attaches hooks to WordPress's runtime to intercept page loads and login eventsgithub.comgithub.com. For instance, it hooks into either the `get_header` or `template_redirect` event (depending on a *Compatibility Mode* setting) to trigger its `jr_ps_force_login()` routine before any page rendersgithub.com. This routine checks if the current user is logged in and authorized; if not, it redirects them to the login screengithub.comgithub.com. The plugin also hooks into other events like `rest_api_init` (to block REST API calls by logged-out users) and various login/logout events to manage redirects and track login stategithub.com.

The main plugin file (`jonradio-private-site.php`) initializes default settings and handles upgrades. It creates two WordPress options for persistence: **`jr_ps_settings`** (for user-configurable settings like "site private" flag, allowed URLs, etc.) and **`jr_ps_internal_settings`** (for internal flags such as installed version, first-run timestamp, etc.). On each run it ensures these options exist and populates any missing default valuesgithub.comgithub.com. The plugin's bootstrap logic then loads the appropriate components: all admin-related modules are required if `is_admin()` (including pages for settings, license info, etc.), and the front-end enforcement logic is only loaded on non-admin requests *and only if* the site is marked private in settingsgithub.com. This design means that if the administrator turns "Private Site" off, the front-end enforcement hook is not added, and the site behaves normally.

Multi-Site Considerations: My Private Site supports WordPress Multisite installations. It can be activated per site or network-wide. When network-activated, it adds a Network Admin settings page that mainly informs the network admin that each site maintains its own privacy settings (there isn't a unified global setting yet)github.comgithub.com. The plugin even works around a WordPress quirk by *surfacing itself* on individual sites' plugin lists: normally network-activated plugins are hidden on sub-sites, but My Private Site injects an entry so that admins of each site see it with a "Settings" linkgithub.comgithub.com. This makes it easier to find the settings page for each site. (It achieves this by manipulating the plugin list table and adding a custom action link, as seen in the `includes/installed-plugins.php` logicgithub.comgithub.com.)

Overall, the architecture is typical for a complex WordPress plugin: a procedural codebase using WP hooks and global functions, split into logical modules. It cleanly separates admin UI from runtime enforcement and uses the options table for state. The flow is: on every page load, if site privacy is active, the `jr_ps_force_login` check runs early and either lets the request through (if user is logged in or the page is allowed) or redirects to logingithub.comgithub.com. After a successful login, a "landing location" logic decides where to send the user (original requested page by default, but customizable). In the dashboard, administrators have multiple settings screens (tabs) to configure all these behaviors.

2. Key Modules & Components

The repository is organized into several modules, each handling a different aspect of the plugin's functionality:

- **Main Plugin Controller:** The root plugin files `jonradio-private-site.php` and `jonradio-private-site-admin.php` are the entry points. The former sets up global plugin data, checks the stored version (to run upgrade routines), initializes default options, and loads either admin or front-end componentsgithub.comgithub.com. The latter (`*-admin.php`) initializes the CMB2 library and includes all the admin sub-modulesgithub.comgithub.com. It also registers a `[privacy]` shortcode for selective content hiding (more below)github.com.
- **Privacy Enforcement (Front-End):** Located in `includes/public.php`, this module implements the logic that actually **forces logins** on protected pages. It defines `jr_ps_force_login()` which runs on every page load (via the hooked actions) to check the conditions for accessgithub.com. Key checks include: Is the user already logged in? (If yes, allow them, unless *Check Role* is enabled and they have no role on this site)github.com. Is the current URL exempted (login page, registration page, home page if configured public, any URL on an "allowed" list)? If any exemption applies, the page is showngithub.comgithub.com. Otherwise, the user is redirected to the login screen (or a custom login URL if configured)github.com. This module also hooks into the **REST API** initialization to block API requests for logged-out users when "REST API Guardian" is enabledgithub.com. In addition, it monitors login events: for example, it sets a flag when a login form is being shown (to avoid redirect loops)github.com, and it filters the login redirect to send users to a chosen landing page after authenticationgithub.com. Overall, *public.php* is the core of the access control mechanism.
- **Admin Settings Pages:** The plugin's settings UI is divided into multiple tabs, each implemented in the `admin/` folder:
    - **"My Private Site" (Main tab):** A welcome/info page (`admin/main.php`). This is the top-level settings page that appears under **Settings** in WP admin. It doesn't have many fields itself aside from a welcome message (and possibly donation links), but it acts as the container for the tabbed interfacegithub.comgithub.com. It uses a WordPress Dashicon lock icon for easy identification in the menugithub.com.
    - **Site Privacy:** (`admin/site-privacy.php`) – **Controls the master privacy switch and basic privacy settings.** This is arguably the most important settings section. It has a checkbox to enable "Make Site Private" which turns the privacy

enforcement on or offgithub.com. When you open this tab, it even displays a bold status indicator ("SITE IS PRIVATE" in green or "SITE IS NOT PRIVATE" in red) for claritygithub.com. This section also offers a *Compatibility Mode* dropdown to handle themes that don't trigger the normal hooks (e.g. certain themes like Elementor require switching to an alternate hook)github.com. Another option here is *Site Privacy Mode* – this is a select field that currently only has one value ("Site Private, Some Pages Public") in the free versiongithub.com. It's essentially a stub for a feature that allows designating specific pages to remain public; implementing multiple modes is left to an add-on (see *Public Pages* add-on below)github.com. Finally, this tab includes the **REST API Guardian** setting – a checkbox to block REST API access by unauthorized usersgithub.com. (When enabled, any REST request from a logged-out user is answered with a 403 error by hooking `rest_pre_dispatch`github.com.) The Site Privacy tab has its own Save buttons for "Privacy Status" and for "REST API Option" which update the `jr_ps_settings` values accordinglygithub.comgithub.com.

- **Landing Page:** (`admin/landing-page.php`) – Controls *where users get redirected after logging in*. This allows the admin to choose the post-login destination (e.g. return to the originally requested URL vs. send everyone to a specific page or the homepage). The code defines options like "Landing Location: *Return to last page* vs *Specific URL*" etc., using fields for a redirect URL if needed and a toggle to omit WordPress's `redirect_to` parameter in certain cases to avoid loopsgithub.com (as noted in the FAQ, WordPress's own redirect logic can conflict if a custom login page is a WordPress pagegithub.com). This tab's settings tie into the `jr_ps_after_login_url()` logic and the `login_redirect` filter seen in the public modulegithub.com.
- **Public Pages (Home Page):** (`admin/public-pages.php`) – Provides an option to leave the site's homepage open to the public even while the rest of the site is private. This is a simple checkbox "Allow site home page to remain accessible without login" (which sets the `excl_home` flag)github.com. In the UI this is labeled *"Public Home Page"*github.com. In the free plugin, that's the extent of per-page granularity: only the homepage can be excluded via settings. The underlying code does support an array of excluded URLs/prefixes (`excl_url` and `excl_url_prefix` in settings)github.comgithub.com, but the interface to manage those is not fully exposed (likely reserved for a premium "Public Pages 2.0" extension). Essentially, the free version gives one checkbox for homepage, whereas the premium add-on allows specifying arbitrary pages or URL patterns to leave publicgithub.com.
- **Selective Content (Shortcodes):** (`admin/selective-content.php`) – Explains and configures the `[privacy]` shortcode feature. My Private Site includes a shortcode `[privacy hide-if="..."]...[/privacy]` which lets you wrap content that should be conditionally hidden based on login statusgithub.comgithub.com. In the free version, the shortcode supports `hide-if="logged-in"` or `hide-if="logged-out"` to hide the enclosed content from logged-in users or from guests, respectivelygithub.comgithub.com. The *Selective Content* admin tab is mostly informational; it provides syntax examples and promotes a more advanced *Selective Content* extension. In fact, the

description on that page mentions capabilities like “hide, scramble, and truncate text based on login or role, and even hide widgets or sidebars”github.com – features which go beyond the basic shortcode and are part of the premium add-ongithub.com. The free plugin’s core implementation for the shortcode is simple: it blanks out the wrapped content if the condition is metgithub.com.

  - **Membership & Registration:** (`admin/membership.php`) – Handles user registration settings. This page has a checkbox to mirror WordPress’s *Anyone can register* option (membership open) and a *Reveal Registration Page* checkboxgithub.com. The latter, when enabled, ensures the standard WP signup page is not hidden by the privacy enforcement (so new users can self-register)github.com. Internally, if “Reveal Registration Page” is checked, the plugin’s front-end logic will allow the `/wp-register.php` or appropriate URL through, as well as BuddyPress registration/activation pages if BuddyPress is detectedgithub.comgithub.com. This tab basically provides a convenient interface to toggle `users_can_register` (it actually updates the core WP option for membership)github.com and the plugin’s own `reveal_registration` settinggithub.com. It ensures administrators don’t accidentally lock out the registration form when the site is private.
  - **Advanced Settings:** (`admin/advanced.php`) – Offers various advanced or edge-case settings. Here admins can configure a **Custom Login Page URL** if they want the plugin to redirect to a specific URL (on or off the site) for logingithub.comgithub.com. There’s also a setting to treat that custom login as on-site or external (so the plugin knows whether to expect a return) and an *“Override Omit `redirect_to`”* option that addresses rare redirect-loop issues by forcibly removing the `?redirect_to=` param under certain conditions (mentioned in changelog and advanced settings). Additionally, the Advanced tab houses the *“Check User Role”* toggle (whether a logged-in user with no role on a site should be denied access – default true) and possibly logging or debugging options. In summary, **Advanced** collects the miscellaneous settings: custom login handling (`custom_login`), homepage redirect rules, role-check override, etc., which more technical users might need.
  - **Add-Ons and Licenses:** (`admin/addons.php` and `admin/licenses.php`) – These pages are mostly informational/marketing. The **Add-ons** tab lists and links to the premium extensions available (e.g. *Public Pages 2.0*, *Tags & Categories*, *Selective Content* extended)github.com, encouraging users to upgrade for those features. The **Licenses** tab likely provides license information for bundled libraries (such as CMB2, which is open source, and others) or a place to enter license keys if any premium features required it. These do not affect functionality in the free plugin but improve user awareness and compliance with open-source licenses.

- **Utility and Legacy Code:** The repository contains a `util/` folder and a `legacy/legacy.php`. The **utilities** provide helper functions and backward compatibility. Notably, there is `includes/common-functions.php` (originally from the plugin’s predecessor *jonradio* plugins) that defines a series of `jr_v#_*` functions – these are versioned utility functions (for string handling, URL parsing, etc.) designed to avoid conflicts if multiple plugins include themgithub.comgithub.com. For example, functions

like `jr_v1_same_url()` (used to compare URLs ignoring minor differences) are defined here and used to check if a given URL should be excluded from login enforcementgithub.comgithub.com. The *legacy.php* likely contains migration code or shims for older plugin versions (ensuring that settings from an old version are mapped to new ones, etc.). Overall, this category ensures older installs upgrade smoothly and provides common helper logic across the plugin. There is also a **telemetry** component: the file `telemetry/deactivate.php` and associated JS/CSS handle the *"deactivation feedback form"*. When an admin attempts to deactivate the plugin, it triggers a modal asking for feedback (why they are deactivating)github.comgithub.com. This feedback mechanism is implemented using the Remodal JS library (included under `library/remodal/` and enqueued in the admin)github.com. It gathers info like plugin version, usage duration, etc., and sends an anonymous survey to the author's servergithub.comgithub.com. This is not directly related to site privacy but is part of code quality/user experience (to help the developer improve the plugin).
- **Included Libraries:** The plugin bundles a few third-party libraries in the `library/` directory. The most significant is **CMB2**, a popular PHP framework for creating WordPress option panels and meta boxes. My Private Site uses CMB2 to generate its settings pages and forms, including the tabbed interface for the multiple subsections. The repository's `library/cmb2/` folder contains this library (version ~2.10)github.com. It's loaded on admin initgithub.com. Another small library included is **Remodal** (for modal dialogs in admin, used by the telemetry feedback). There is also a `library/node-uuid` – likely a UUID generation library (perhaps used to generate a unique ID for the site or feedback, although it's not immediately invoked in the core code we reviewed). These libraries are included to avoid external dependencies and are initialized as needed.

In summary, **My Private Site** is composed of modular components for each feature area: login enforcement on the front-end, and a suite of admin pages for configuration (Privacy, Landing, Registration, etc.), all tied together by central bootstrap logic. The use of CMB2 means much of the settings UI code is about defining fields and letting the library handle rendering and saving. This keeps the code organized and relatively easy to maintain or extend (as evidenced by the presence of premium add-on hooks and upgrade placeholders in the UI).

3. Tech Stack & Primary Technologies

My Private Site is built entirely on **WordPress's platform**, using **PHP** as the programming language. Key elements of its tech stack include:

- **WordPress Core APIs:** The plugin heavily uses WordPress hook APIs (`add_action`, `add_filter`) to intercept page loads, login events, and admin page initializationgithub.com. It relies on WordPress functions like `is_user_logged_in()`, `wp_login_url()`, `wp_redirect()`, and option management (get/update_option) to implement its featuresgithub.comgithub.com. It also checks multisite-specific functions (`is_multisite()`, `is_user_member_of_blog()`, etc.) for network supportgithub.com. All integration is done the "WordPress way," ensuring compatibility with the ecosystem.
- **PHP OOP/Functional Mix:** The code is largely procedural (function-based), with a few global variables used to share state (e.g. `$jr_ps_is_login` flag, `$jr_ps_plugin_data`

array holding plugin metadata)github.comgithub.com. There isn't a singular class for the plugin, which is common for legacy WP plugins. Instead, it uses unique prefixes (`jr_ps_`) to avoid naming collisions. It's structured for WordPress 4.0+ (as specified in the readme), but even contains some fallbacks for older PHP versions (e.g. multibyte string function exists checks in `common-functions.php`)github.com.

- **CMB2 (Custom Meta Boxes 2):** This is an external PHP framework bundled with the plugin for creating option pages easily. My Private Site uses CMB2 to generate its settings pages as an **"options-page"** type with tab support. Each settings tab (Privacy, Advanced, etc.) is defined via CMB2's API (fields, boxes, etc.), and CMB2 handles rendering them in the WordPress admin. The plugin includes and initializes CMB2 on admin loadgithub.com and uses features like the tab navigation functionality provided by CMB2 (through a `display_cb` callback and grouping the pages in a tab set)github.com. Using CMB2 greatly simplifies the creation of forms and handles things like nonces and field sanitization. (We can see CMB2 usage in the code, for example calling `new_cmb2_box()` to create a settings page and `$box->add_field()` to add each field controlgithub.comgithub.com.)
- **Web Technologies (HTML, JS, CSS):** The plugin outputs HTML mainly through WordPress admin pages and uses the WordPress style where possible. For custom needs, it has an `adminstyles.css` for styling its admin screens (e.g. making certain notices or layouts look nice). On the JavaScript side, the plugin doesn't have a large custom script for functionality except for the telemetry modal. It enqueues a minified **Remodal.js** library and corresponding CSS theme for the feedback dialoggithub.com, plus a small custom script (`telemetry/js/deactivate.js`) to drive the survey form behavior on the plugins pagegithub.com. This script likely uses jQuery (since WP admin loads it) to open the modal, collect selected reasons, and send an AJAX request to the telemetry URLgithub.comgithub.com. All UI strings in that feedback form are localized for i18n via `wp_localize_script`github.com. Speaking of localization, the repository includes `.po` files for various languages (suggesting the plugin is translation-ready).
- **Platform Compatibility:** The plugin declares compatibility up to WordPress 6.8 and requires at least WP 4.0github.com. It uses PHP 5.6+ syntax and is fully functional on PHP 7.x+ (the codebase shows attention to PHP 7 compatibility, such as avoiding deprecated constructs and addressing PHP7 warnings as noted in CMB2 updates). The presence of PHPCS comments in code (to ignore certain false positives) indicates the developers run code sniffers for WordPress coding standardsgithub.com, which is a good practice.

Overall, My Private Site's tech stack is a fairly standard WordPress plugin stack: **PHP + WordPress**, augmented by a *meta-box library (CMB2)* for admin UI, some *JavaScript/CSS* for niceties in the dashboard (modal feedback form), and standard web tech for output. There is no external server component except the optional telemetry pings. Everything executes within the WordPress plugin environment. This choice of technologies makes the plugin easy to install and use without any extra setup – it's self-contained in the WordPress ecosystem.

4. Notable Open Issues and Upcoming Changes

The repository does not list any critical open bug issues at the moment – the plugin is mature (version 3.1.1) and most known bugs have been addressed in recent updates (as evidenced by the changelog fixes for Elementor compatibility, security improvements, etc.)github.comgithub.com. However, there are a few **enhancements and outstanding ideas** worth noting for future development:

- **Network-Wide Settings Feature:** Currently, when used on a Multisite network, each site has its own privacy settings. The plugin's Network Admin page explicitly informs admins that a single unified privacy setting for the whole network is not yet available. It invites users to contact the author if they want this feature, indicating it's a planned (but demand-driven) enhancementgithub.com. This means a future version might introduce a global "make all sites private" toggle or a central management interface if enough interest is shown. This is something to watch if you run a network – for now, you must configure each site individually.
- **"Public Pages" Mode Expansion:** The UI includes a *Site Privacy Mode* dropdown (under Site Privacy settings) which hints at two modes: the ability to run the site in private mode (with exceptions) or in public mode with only certain pages protectedgithub.com. In the free version, only the first mode is implemented ("Site Private, Some Pages Public")github.com. The second scenario (site generally public, but some pages private) is actually the inverse functionality – and it's being offered via a premium addon (Public Pages 2.0). The presence of this field and the description text shows the developer's plan to support both privacy configurationsgithub.com. It's notable that as of now, the core plugin itself doesn't let you mark specific pages as private while the site is public – but the groundwork is there. If you need that capability, the *Public Pages* extension would be the solution, and we might see more integration of it or a basic version in future updates. In short, keep an eye on this feature as it evolves; it's a logical extension of the plugin's functionality.
- **Compatibility and Bug Fixes:** My Private Site has had a history of quickly addressing compatibility issues with other themes and plugins. For example, earlier issues with Elementor (a popular page builder) were resolved by introducing the Compatibility Mode settinggithub.com. The only known plugin conflict listed is with *A5 Custom Login*, due to overlapping login modificationsgithub.com. Given that, any new conflicts that arise (for instance, if a future WordPress core update changes the login flow or a major plugin introduces a breaking change) would likely be tracked and fixed promptly. There are no open pull requests currently on the repository, and recent commits have focused on minor tweaks (UI wording improvements in 3.1.1, and adding the REST API blocking in 3.1.0)github.com. This suggests the plugin is stable, and development is now centered on incremental improvements and supporting add-ons rather than large changes.
- **User Feedback & Telemetry:** The inclusion of the deactivation feedback form indicates the author is gathering user input on why they disable the plugin. This might not directly correspond to a GitHub issue, but it means common pain points or requests from users could translate into future changes. For example, if multiple users deactivate citing "needed to allow some pages public," that reinforces the development of the Public Pages feature. It's worth noting that support has moved off the WordPress.org forums to the developer's own sitegithub.com. So "issues" are likely handled via that ticket system.

Any notable tickets (such as a bug report or feature request) would be addressed in upcoming releases rather than tracked publicly on GitHub.

In summary, **no major unresolved issues** are present publicly, but the roadmap hints at:

- A possible **network-wide setting** if demand growsgithub.com.
- More flexible **public/private page granularity** (currently via add-ons) being a focusgithub.com.
- Ongoing maintenance for **compatibility with new WP versions** (the plugin has been kept up-to-date for WP 6.x and will likely continue that; e.g., a minor update ensured compatibility with WordPress 6.4github.com).

For a user of the plugin, it's recommended to keep it updated to get these improvements. If you need a feature that isn't in core (like private-by-role or other nuanced control), check the add-ons or contact the developer – it might be in the works if enough users ask.

5. Suggestions to Improve Code Quality

While the codebase is functional and has proven itself over years, there are a few opportunities to enhance its quality and maintainability:

- **Modernize and Refactor the Code Structure:** The plugin could benefit from using a more modern, object-oriented approach. Currently, it relies on many global functions and variables (`jr_ps_*` namespaced) spread across files. Encapsulating related functionality into classes or at least namespaced functions would improve readability and prevent any potential naming collisions. For instance, a single `My_Private_Site` class could hold initialization logic, and separate classes or files for Admin, Frontend Enforcement, etc., could limit the reliance on global state. This would make the code easier to navigate and extend. As part of this modernization, the team could remove remnants of very old compatibility layers. The code still contains checks for WordPress 3.0 era features (e.g. conditionally calling `is_network_admin()` only if it exists) which are not necessary nowgithub.com. Dropping such legacy conditions and requiring a reasonable minimum WP version would simplify the logic and reduce dead code. Adopting PHP namespaces or at least consistent class prefixes for new code would also avoid the need for the custom versioned function approach in `common-functions.php` (which was clever for sharing code in the past, but is less needed if structured into a single plugin scope). In short, a refactoring pass focusing on organization (perhaps grouping functions into classes/modules) and eliminating outdated shims would enhance the code quality without changing functionality.
- **Improve Consistency and Maintainability:** There are some minor inconsistencies and areas to streamline:
  - o *Admin UI code duplication:* Each admin tab file repeats a lot of boilerplate (setting up essentially similar CMB2 boxes). Abstracting some of this (for example, a helper function to create the box with common args, or using loops for repetitive field sets) could reduce repetition. It's minor, but would ease future

changes (like if they wanted to alter how tabs are rendered, currently they must do it in each file).

- *Settings handling:* The plugin uses a single options array for many settings, which is fine, but when new settings are introduced (e.g. `private_api` for REST), they should be added to the defaults initialization. Ensuring that `jr_ps_init_settings` always knows about new keys will prevent undefined indexes. Currently, `private_api` wasn't in the hard-coded defaults array for `jr_ps_settings` in the main file (perhaps because it's new in 3.1.0), meaning it only gets created when saved via the form. It would be safer to include it with a default (`false`) so that all options are initialized consistently. Little things like this tighten up the reliability of the code.
- *Code comments and documentation:* Generally the plugin is well-commented (each function has a description, and important warnings are noted). Continuing this practice and maybe adding a short developer documentation (for example, listing the plugin's custom hooks like `my_private_site_public_check_access_by_page` filter) would help third-party developers or future maintainers. The changelog shows that new filters were added for granular controlgithub.com; documenting these in the code or readme would be beneficial.
- *Remove commented-out or unused code:* There are a few sections that appear to be leftover (for example, an commented filter addition for plugin action links in installed-plugins.php, or legacy code that's now bypassed by new logic). Cleaning these up will reduce confusion. For instance, the "plugin_action_links" for the settings link is partly commented out and re-implemented in a different way for network vs single sitesgithub.comgithub.com. This could be refactored to a clearer single approach now that WordPress is more uniform, making the intent more obvious.

By implementing these suggestions, the plugin's codebase will become more robust and easier to work with. It's a mature project, so any refactoring should be done carefully to avoid breaking compatibility – but incremental improvements (like phasing out old hacks and using current best practices) will pay off in the long run. The good news is the plugin already follows many WP conventions and has a solid structure; the above steps would further align it with modern WordPress coding standards and ensure it remains maintainable for years to come.

Overall, **My Private Site** is a well-architected plugin that does its job effectively. The recommendations above aim to make the developer's life easier (and the software more future-proof) without altering the end-user experience. Keeping the code lean, modern, and well-organized will help the project continue to thrive alongside WordPress's evolutiongithub.comgithub.com.