



metamint

Helpdesk

Addon

Developer Documentation

Developer Documentation — Integration & Productivity Tools

Plugin Version: 1.21.0 | Text Domain: metamint-helpdesk-addon

A complete technical reference for extending, maintaining, and integrating with the Metamint Helpdesk Addon plugin

Plugin: mmhd-addon | Version: 1.21.0

Requires: Metamint Helpdesk core plugin (mmhd.php) — WordPress

Publisher: CodePixelz Media

1. Introduction.....	5
1.1 What is Metamint Helpdesk Addon?.....	5
1.2 Purpose of this document.....	5
1.3 Audience.....	5
2. System Architecture.....	5
2.1 Two-plugin model.....	5
2.2 Dependency check.....	6
2.3 Bootstrap sequence (mmhd-addon.php).....	6
3. The Addon Manager.....	7
3.1 Construction flow.....	7
3.2 Addon registry schema.....	7
3.3 Bundled addons.....	7
3.4 Public API (MMHD_Addon_Manager).....	8
3.5 REST endpoint.....	8
4. Licensing System.....	9
4.1 License states.....	9
4.2 Storage.....	9
4.3 Remote API.....	9
4.4 Filters.....	10
4.5 Scheduled check.....	10
5. Auto-Update System.....	11
5.1 Update discovery.....	11
5.2 Download gate.....	11
5.3 Every admin_init cache clears.....	11
6. Plugin Constants Reference.....	12
7. Hooks & Filters Reference.....	12
7.1 Hooks provided by Metamint Helpdesk core (consumed by addons).....	12
7.2 Hooks provided by Metamint Helpdesk Addon (for addon and third-party developers).....	13
8. Building a Custom Addon.....	14
8.1 Folder structure convention.....	14
8.2 addon.json schema.....	14
8.3 Registering the addon.....	15
8.4 The bootstrap file.....	15
8.5 Adding an AJAX handler.....	16
8.6 Gating on the shared license.....	17
9. AJAX API Reference.....	18
9.1 Core addon-manager actions.....	18
9.2 Per-addon AJAX actions.....	18
10. REST API Reference.....	22
10.1 Metamint Helpdesk Addon core.....	22
10.2 AI Bot (mmhd-bot/v1).....	22

10.3 Zapier (mmhd-zapier/v1).....	22
10.3.1 Create Ticket.....	23
10.3.2 Add Reply.....	24
10.3.3 Update Ticket.....	24
10.3.4 Create User.....	25
10.3.5 Delete User.....	25
10.3.6 Delete Ticket.....	26
10.3.7 List Tickets.....	26
10.3.8 List Replies.....	27
10.3.9 List Users.....	27
10.3.10 List Deleted Tickets.....	28
10.3.11 List Deleted Users.....	28
10.3.12 List Categories.....	29
10.3.13 List Priorities.....	29
10.4 Error Responses.....	29
10.5 — Rating & Review (mmhd-rating/v1).....	30
11. Addon-by-Addon Reference.....	31
11.1 Slack Integration.....	31
11.2 Discord Integration.....	32
11.3 Signature.....	33
11.4 Canned Replies (Saved Replies).....	34
11.5 AI Bot.....	34
11.6 Filter Emails.....	35
11.7 WooCommerce.....	36
11.8 Multi-Inbox.....	36
11.9 WhatsApp Integration.....	37
11.10 Zapier Integration.....	38
11.11 — Rating & Review.....	41
12. Database & Storage Reference.....	43
12.1 Custom database table.....	43
12.2 WordPress options reference.....	44
13. Security Guidelines.....	47
14. Coding Standards & Tooling.....	48
14.1 Autoloading.....	48
14.2 Coding standard.....	48
14.3 Function naming convention.....	48
14.4 Text domain.....	48
Appendix A: addon.json Field Reference.....	49
Appendix B: Quick Reference Cheat Sheet.....	49

1. Introduction

1.1 What is Metamint Helpdesk Addon?

Metamint Helpdesk Addon is a companion WordPress plugin to the core Metamint Helpdesk helpdesk/support-ticket plugin. It is a hard-dependent, license-gated "addon pack" that bundles eleven optional integration modules (Slack, Discord, WhatsApp, Zapier, WooCommerce, AI Bot, Multi-Inbox, Filter Emails, Signatures, Canned Replies, Rating & Review) behind a single centrally-managed activation UI (Metamint Helpdesk → Addons).

Each bundled module is called an "addon." Addons are self-contained folders under `/addons/{slug}/` that plug into Metamint Helpdesk core through a documented set of WordPress hooks, and into the Metamint Helpdesk Addon shell through a small, consistent bootstrap convention.

1.2 Purpose of this document

This document is the developer reference for the Metamint Helpdesk Addon codebase. It covers:

- The plugin architecture and bootstrap sequence
- The Addon Manager and addon lifecycle (register, load, activate, deactivate)
- The licensing system that gates addon functionality
- The auto-update mechanism
- Every hook, filter, AJAX action and REST route exposed by Metamint Helpdesk core and by Metamint Helpdesk Addon
- A step-by-step guide to building a new custom addon
- A deep-dive reference for each of the eleven bundled addons
- Database tables and WordPress options used for persistence
- Security and coding-standard conventions

1.3 Audience

WordPress plugin developers who need to build a new Metamint Helpdesk addon, extend an existing addon, integrate a third-party service, or simply understand how the two plugins (Metamint Helpdesk and Metamint Helpdesk Addon) talk to each other.

2. System Architecture

2.1 Two-plugin model

Metamint Helpdesk Addon is a strict dependency of the Metamint Helpdesk core plugin. It will not run — and will self-deactivate with an admin notice — if Metamint Helpdesk core is not active.

Plugin	Role	Main file
metamint-helpdesk	Core helpdesk engine: tickets, replies, categories, dashboard shell, settings-tab framework, REST/AJAX for ticket operations.	mmhd.php
metamint-helpdesk-addon	Optional-feature layer: bundles 11 addons, license gating, auto-update, addon activation UI.	mmhd-addon.php

2.2 Dependency check

`mmhd_addon_is_mmhd_active()` checks, in order: runtime constants/functions already loaded (`MMHD_PLUGIN_PATH`, `mmhd_dashboard_callback()`), `is_plugin_active('metamint-helpdesk/mmhd.php')`, network activation, and finally a fallback scan of `active_plugins` for any plugin file named `mmhd.php` (so a renamed folder is still detected).

Note: `mmhd_addon_check_dependencies()` is called on every `plugins_loaded` (priority 10). If the Metamint Helpdesk core is missing, the addon plugin calls `deactivate_plugins()` on itself and prints an `admin_notices` warning — it never runs partially.

2.3 Bootstrap sequence (mmhd-addon.php)

Hook / priority	What happens
File load	<code>vendor/autoload_packages.php</code> (Jetpack autoloader) is required first, then plugin constants are defined.
<code>admin_init</code> (priority 1)	Clears on every <code>admin_init</code> : deletes the <code>mmhd_addon_remote_version</code> transient and the <code>update_plugins</code> site transient (no option flag; runs each page load).
<code>plugins_loaded</code> (priority 5)	"Always-on" hooks that must run even when the license gate would otherwise block loading (e.g. AJAX/REST requests). Currently used to require <code>addons/whatsapp/includes/whatsapp-reply-hook.php</code> unconditionally.
<code>plugins_loaded</code> (priority 10)	<code>mmhd_addon_init()</code> — the main bootstrap: checks dependencies, boots the license service, requires <code>includes/addon-manager.php</code> , instantiates the Addon Manager, and (if <code>is_admin()</code>) requires <code>admin/mmhd-addon-admin.php</code> .
<code>init</code>	Loads the <code>mmhd-addon</code> text domain for translations.
<code>register_activation_hook</code>	<code>mmhd_addon_activate()</code> : re-checks the Metamint Helpdesk dependency (<code>wp_die</code> if missing), sets <code>mmhd_addon_activated=true</code> , schedules the daily license-check cron, runs an immediate license check, and re-schedules the Slack reminder cron if it was previously enabled.
<code>register_deactivation_hook</code>	<code>mmhd_addon_deactivate()</code> : deletes the <code>mmhd_addon_activated</code> option and clears the daily license-check and Slack reminder cron hooks.

Note: Because the `addon-manager` singleton is constructed inside `mmhd_addon_init()` (priority 10), any code that needs to inspect the addon registry must run at `plugins_loaded` priority ≥ 10 , or later (`init`, `admin_menu`, etc.).

3. The Addon Manager

Class: `MMHD_Addon_Manager` — `includes/addon-manager.php`. Implemented as a classic singleton accessed through the `mmhd_addon_manager()` helper function.

3.1 Construction flow

The private constructor runs three steps in order:

- `register_addons()` — populates the in-memory `$addons` registry (see 3.2).
- `load_active_addons()` — iterates the registry and `require_once()`s the main file of every addon whose stored status option is 'active', but only if the license is currently valid.
- Registers the REST route `mmhd-addon/v1/status` on `rest_api_init`.

3.2 Addon registry schema

`register_addons()` currently builds a hard-coded PHP array (a commented-out auto-discovery variant that scans `addons/*/addon.json` with `glob()` is kept in the file for reference/future use). Each entry has the following shape:

Key	Type	Description
name	string (i18n)	Human-readable addon name shown in the Addons grid.
description	string (i18n)	One-line description shown under the name.
version	string	Addon version (independent of the parent plugin version).
icon	string	Font Awesome class (e.g. <code>fab fa-slack</code>) or a relative path to an image. The registry icon for WooCommerce is the Font Awesome class <code>fas fa-cart-shopping</code> ; an SVG logo exists only in <code>addon.json/assets</code> , not in the registry.
icon_bg / icon_color	string	Tailwind-style color tokens used by the admin UI for the icon badge.
path	string	Absolute filesystem path to the addon folder, e.g. <code>MMHD_ADDON_PLUGIN_PATH . 'addons/slack/'</code> .
main_file	string	Filename of the addon's bootstrap file, relative to path.
settings_tab	string (optional)	Slug is used when the addon registers a tab via the <code>mmhd_settings_tabs</code> filter.
status	'active' 'inactive'	Read from the WordPress option <code>mmhd_addon_{slug}_status</code> ; defaults differ per addon (Slack, Zapier, and WhatsApp default to active, the rest default to inactive).

3.3 Bundled addons

Slug	Name	Default status	Settings tab
slack	Slack Integration	active	slack

Slug	Name	Default status	Settings tab
discord	Discord Integration	inactive	discord
signature	Signature	inactive	—
saved_replies	Canned Replies	inactive	—
ai_bot	AI Bot	inactive	ai_bot
filter_emails	Filter Emails	inactive	filter-emails
rating_review	Rating & Review	inactive	rating-review
woocommerce	WooCommerce	inactive	—
multi_inbox	Multi-Inbox	inactive	imap
whatsapp	WhatsApp Integration	active	whatsapp
zapier	Zapier Integration	active	zapier

3.4 Public API (MMHD_Addon_Manager)

Method	Purpose
get_instance()	Static accessor for the singleton (also exposed via the mmhd_addon_manager() function).
get_addons()	Returns the full registry array.
get_addon(\$key)	Returns a single addon's config array, or null.
is_active(\$key)	Returns true if the addon's stored status is 'active'.
activate(\$key)	Requires a valid license; updates the status option to 'active', calls load_addon(), then fires do_action('mmhd_addon_after_activate', \$key, \$addon).
deactivate(\$key)	Calls mmhd_addon_{key}_deactivate() if it exists, fires do_action('mmhd_addon_before_deactivate', ...), sets status to 'inactive', then fires do_action('mmhd_addon_after_deactivate', ...).
deactivate_all_active()	Deactivates every currently-active addon; used by the license service when a license becomes invalid.
load_addon(\$key) (private)	require_once()s the addon's main file, then calls mmhd_addon_{key}_init() if that function exists.

3.5 REST endpoint

Method	Route	Auth	Response
GET	/wp-json/mmhd-addon/v1/status	Public (permission_callback returns true)	{ activated: bool, status: 'active' 'inactive' } based on the mmhd_addon_activated option.

4. Licensing System

Class: MMHD_Addon_License_Service — includes/class-mmhd-addon-license-service.php.

Gatekeeper for whether any bundled addon is allowed to load at all. Accessed through the mmhd_addon_license_service() singleton helper.

4.1 License states

Constant	Value	Meaning
STATE_INACTIVE	inactive	No license activated on this site (default).
STATE_ACTIVE	active	Valid, currently-usable license.
STATE_EXPIRED	expired	License existed but its term has lapsed.
STATE_SUSPENDED	suspended	License temporarily disabled server-side.
STATE_REVOKED	revoked	License permanently revoked.

Note: mmhd_addon_has_valid_license() (used everywhere addons check gating) is true only when the state is STATE_ACTIVE. Any other state causes MMHD_Addon_Manager::load_active_addons() to skip loading addon files entirely, and deactivate_all_addons_if_invalid() will flip every addon's stored status back to inactive.

4.2 Storage

Option	Contents
mmhd_addon_license_settings	{ email, key } — the credentials currently saved for this site.
mmhd_addon_license_state	One of the five state strings above.
mmhd_addon_license_meta	{ licenseId, orderId, expiresAt, checkedAt } — cached metadata from the last successful API call.

4.3 Remote API

Base URL (overridable): <https://metamintapps.com/wp-json/mmhd-licensing/v1/> — filterable via `mmhd_addon_license_api_base`. All requests are POST with a Bearer token (the license key) and a JSON body of the shape `{ name, param: { email, domain, siteFingerprint } }`.

Method	Endpoint (relative)	Purpose
<code>activate(\$email, \$key)</code>	<code>license/activate</code>	Activates a license for this site; on success stores settings + <code>STATE_ACTIVE</code> + meta.
<code>deactivate()</code>	<code>license/deactivate</code>	Releases the site's activation slot; clears settings and sets <code>STATE_INACTIVE</code> .
<code>check_status(\$force = false)</code>	<code>license/status</code>	Refreshes state. Cached for 1 hour unless <code>\$force</code> is true (the daily cron always forces).
<code>check_status_with(\$email, \$key)</code>	<code>license/status</code>	Same as above but checks arbitrary, unsaved credentials (used by the "Check" button before saving).

`get_site_fingerprint()` derives a stable, non-reversible per-site identifier:
`sha256(host|admin_email|wp_salt('auth'))`, used by the licensing API to bind an activation to a specific site.

4.4 Filters

Filter	Default	Purpose
<code>mmhd_addon_license_api_base</code>	<code>DEFAULT_API_BASE</code> constant	Override the licensing API base URL (e.g. for a staging licensing server).
<code>mmhd_addon_license_sslverify</code>	<code>true</code>	Defaults to true. Override via the <code>mmhd_addon_license_sslverify</code> filter to disable SSL verification for the licensing API only (e.g. a local licensing server).

4.5 Scheduled check

A daily cron event, `mmhd_addon_daily_license_check`, is scheduled on activation (first run 5 minutes later, then every 'daily' interval) and calls `check_status(true)` to force a live re-validation even if the 1-hour cache would otherwise be used.

5. Auto-Update System

File: includes/plugin-updater.php. Metamint Helpdesk Addon ships its own lightweight updater instead of relying on WordPress.org, because it is a commercially licensed plugin.

5.1 Update discovery

Hooked on `pre_set_site_transient_update_plugins`. `mmhd_addon_check_for_updates()` calls `mmhd_addon_fetch_version_data()` against `MMHD_ADDON_VERSION_API` (<https://metamintapps.com/wp-json/mmhd-api/v1/update-version>), compares the returned version to `MMHD_ADDON_VERSION` with `version_compare()`, and if newer, injects a response entry into the update transient pointing package at `MMHD_ADDON_ZIP_URL`.

5.2 Download gate

Hooked on `upgrader_pre_download`. `mmhd_addon_validate_license_before_download()` intercepts WordPress's own attempt to download the plugin ZIP and validates the license server-side before allowing the download to proceed — an unlicensed site cannot pull a plugin update even if it discovers one is available.

5.3 Every `admin_init` cache clears

On every `admin_init` (priority 1) the plugin deletes the `mmhd_addon_remote_version` transient and the `update_plugins` site transient, guaranteeing the update check re-runs rather than serving a stale cached "no update" result while this version of the code is deployed.

In addition to the remote-update path above, the updater supports a same-site/self-hosted path: if a local ZIP exists at `mmhd-updates/metamint-helpdesk-addon.zip` (next to `wp-content/`), it looks up the site's license key and it's served from inside the `upgrader_pre_download` hook (`mmhd_addon_validate_license_before_download`). `plugins_api` is only the plugin-details popup instead of hitting the remote API, via a `plugins_api` filter, only `mmhd_addon_remote_version` is cleared on `upgrader_process_complete`; `update_plugins` is cleared via `admin_init` (priority 1). This path actively reads `mmhd_io_licensing_secret` (see Section 12.2) — the option is not merely reserved.

6. Plugin Constants Reference

Constant	Value / meaning
MMHD_ADDON_VERSION	'1.11.9' — current plugin version.
MMHD_ADDON_PLUGIN_PATH	plugin_dir_path(__FILE__) — absolute server path to the plugin root.
MMHD_ADDON_PLUGIN_URL	plugin_dir_url(__FILE__) — public URL to the plugin root.
MMHD_ADDON_SLUG	'mmhd-addon' — plugin slug.
MMHD_ADDON_FILE	__FILE__ of mmhd-addon.php.
MMHD_ADDON_VERSION_API	Remote endpoint used by the updater to check the latest published version.
MMHD_ADDON_ZIP_URL	Remote endpoint that serves the plugin ZIP for updates.
MMHD_ADDON_LICENSE_API	The '/validate-license' endpoint it targets is legacy

SSL verification for remote calls can be relaxed per-site via the `mmhd_addon_license_sslverify` filter (defaults to true).

7. Hooks & Filters Reference

7.1 Hooks provided by Metamint Helpdesk core (consumed by addons)

These actions/filters are fired by the Metamint Helpdesk core plugin (mmhd.php) and are the primary integration surface every addon uses to react to ticket events or inject UI.

Hook	Type	Fired when / used for
mmhd_after_ticket_created	action	A new support ticket has been created (from email, dashboard, or REST). Used by Zapier triggers; Slack and Discord instead hook <code>save_post_mmhd_ticket</code> , and WhatsApp does not hook this action at all.
mmhd_after_reply_added	action	A reply has been added to a ticket. Used by Slack/Discord thread updates, Zapier's New Reply trigger.
mmhd_after_ticket_closed	action	A ticket transitions to closed. Used for notification addons.
mmhd_after_ticket_status_change	action	Any status transition (not only closed). Used by Zapier's Status Changed trigger and reminder logic.
mmhd_before_reply_editor	action	Fires just before the agent reply editor renders in the dashboard — an injection point for UI above the editor.
mmhd_after_reply_editor	action	Fires just after the reply editor — used by the Signature addon to render its "Include signature" checkbox.

Hook	Type	Fired when / used for
mmhd_settings_content	action	Fires inside the Settings screen where a tab's body should render — every addon with a settings_tab hooks here to output its settings UI.
mmhd_settings_tabs	filter	Filters the associative array of Settings tab labels/icons — addons add their own tab key here (e.g. 'slack' => '<i ...> Slack').
mmhd_before_create_ticket_from_email	filter	Filters/short-circuits ticket creation from an incoming email — used by the Filter Emails addon to block blacklisted senders and by Multi-Inbox to tag the source inbox.
mmhd_frontend_dashboard_menu_items	filter	Filters menu items shown on the customer-facing frontend dashboard.

7.2 Hooks provided by Metamint Helpdesk Addon (for addon and third-party developers)

Hook	Type	Fired when
mmhd_addon_after_activate	action(\$addon_key, \$addon_config)	Immediately after MMHD_Addon_Manager::activate() flips an addon to active and loads it.
mmhd_addon_before_deactivate	action(\$addon_key, \$addon_config)	Immediately before an addon's status option is set to inactive — the last chance to read still-live addon state.
mmhd_addon_after_deactivate	action(\$addon_key, \$addon_config)	Immediately after an addon has been deactivated.

Note: Each addon can also expose its own function-name convention as a lightweight lifecycle hook: mmhd_addon_{slug}_init() (called on load) and mmhd_addon_{slug}_deactivate() (called on deactivation) — see Section 8.

8. Building a Custom Addon

This section is a step-by-step guide to adding a brand-new addon to the bundle, following the exact conventions the existing eleven addons use.

8.1 Folder structure convention

```
addons/{your-slug}/
  addon.json                # metadata (name, version, icon,
  settings_tab...)
  {your-slug}-addon.php      # bootstrap file (required, name must match
  addon.json main_file)
  includes/
    {your-slug}-handlers.php # AJAX handlers / business logic
    {your-slug}-settings-ui.php # HTML for the Settings tab (if any)
  assets/
    js/{your-slug}-integration.js
    css/{your-slug}-styles.css
```

8.2 addon.json schema

```
{
  "slug": "your_slug",      // underscore form, used in option/function names
  "name": "Your Addon Name",
  "description": "One-line description shown in the Addons grid.",
  "version": "1.0.0",
  "icon": "fas fa-plug",     // Font Awesome class OR relative image path
  "icon_bg": "blue-100",
  "icon_color": "blue-600",
  "main_file": "your-slug-addon.php",
  "settings_tab": "your-slug", // omit if the addon has no Settings tab
  "text_domain": "metamint-helpdesk-addon"
}
```

Important: `addon.json` is currently metadata-only documentation; the live registry is the hard-coded array inside `MMHD_Addon_Manager::register_addons()`. A commented-out `glob()`-based loader that reads `addon.json` automatically already exists in `includes/addon-manager.php` — to ship a new addon today you must add a matching entry to that array as well as create `addon.json`.

8.3 Registering the addon

Add an entry to the `$this->addons` array inside `MMHD_Addon_Manager::register_addons()` (includes/addon-manager.php), following the schema in Section 3.2:

```
'your_slug' => [
    'name'      => __('Your Addon Name', 'metamint-helpdesk-addon'),
    'description' => __('One-line description.',
'metamint-helpdesk-addon'),
    'version'   => '1.0.0',
    'icon'      => 'fas fa-plug',
    'icon_bg'   => 'blue-100',
    'icon_color' => 'blue-600',
    'path'      => MMHD_ADDON_PLUGIN_PATH . 'addons/your-slug/',
    'main_file' => 'your-slug-addon.php',
    'settings_tab' => 'your-slug', // omit if not needed
    'status'    => get_option('mmhd_addon_your_slug_status',
'inactive'),
],
```

8.4 The bootstrap file

Every addon's main file follows the same shape: an `ABSPATH` guard, a `mmhd_addon_{slug}_init()` function that wires up hooks, and (optionally) a `mmhd_addon_{slug}_deactivate()` cleanup function. The Addon Manager calls `init()` automatically after `require_once()`-ing the file, and calls `deactivate()` automatically when the addon is turned off.

```
<?php
if ( ! defined('ABSPATH') ) { exit; }
function mmhd_addon_your_slug_init() {
    require_once __DIR__ . '/includes/your-slug-handlers.php';
    // Only enqueue assets on the Metamint Helpdesk dashboard screen
    add_action('admin_enqueue_scripts', 'mmhd_addon_your_slug_enqueue_scripts');
    // Register a Settings tab (optional)
    add_filter('mmhd_settings_tabs', 'mmhd_addon_your_slug_add_settings_tab');
    add_action('mmhd_settings_content', 'mmhd_addon_your_slug_render_settings');
    // React to core ticket events (optional)
    add_action('mmhd_after_ticket_created', 'mmhd_addon_your_slug_on_new_ticket');
}
function mmhd_addon_your_slug_add_settings_tab($tabs) {
    if (! isset($tabs['your-slug'])) {
        $tabs['your-slug'] = '<i class="fas fa-plug mr-1"></i>' . __('Your Addon',
'metamint-helpdesk-addon');
    }
    return $tabs;
}
function mmhd_addon_your_slug_render_settings() {
```

```

    include __DIR__ . '/includes/your-slug-settings-ui.php';
}
function mmhd_addon_your_slug_enqueue_scripts($hook) {
    if (isset($_GET['page']) && $_GET['page'] === 'mmhd-dashboard') {
        wp_enqueue_script(
            'mmhd-your-slug-integration',
            plugin_dir_url(__FILE__) .
'assets/js/your-slug-integration.js',
            ['jquery'],
            MMHD_ADDON_VERSION,
            true
        );
    }
}
function mmhd_addon_your_slug_on_new_ticket($ticket_id) {
    // ... your integration logic
}
function mmhd_addon_your_slug_deactivate() {
    // Cleanup transients / scheduled events on deactivation
    delete_transient('mmhd_your_slug_cache');
}
mmhd_addon_your_slug_init();

```

Note: Notice the final line: `mmhd_addon_your_slug_init()` is called directly at the bottom of the file. This mirrors most bundled addons (`signature-addon.php`, `slack-addon.php`, `whatsapp-addon.php`, etc.) — the Addon Manager's `load_addon()` also tries to call the `init` function again by convention, but calling it unconditionally at the end of the file guarantees the addon initializes correctly even if it is required directly.

8.5 Adding an AJAX handler

Handlers live in `includes/{slug}-handlers.php`. Only the 5 core `addon-manager/license` AJAX actions (Section 9.1) use the shared admin-guard pattern defined in `admin/mmhd-addon-admin.php`; every bundled addon instead verifies its own per-action nonce (e.g. `mmhd_slack_credentials_action`, `mmhd_discord_save_bot_token`, `mmhd_signature_save_settings`, `mmhd_saved_replies_create`). The shared guard is shown below as one valid option for a new addon's handlers:

```

add_action('wp_ajax_mmhd_your_slug_save_settings', 'mmhd_addon_your_slug_save_settings');
function mmhd_addon_your_slug_save_settings() {
    mmhd_addon_admin_guard(); // verifies nonce 'mmhd_addon_nonce' + capability
'mmhd_access_dashboard'
    $value = isset($_POST['value']) ?
sanitize_text_field(wp_unslash($_POST['value'])) : '';

```

```
update_option('mmhd_your_slug_settings', $value);  
wp_send_json_success(['message' => 'Saved.']);  
}
```

Note: `mmhd_addon_admin_guard()` expects the JS caller to POST a nonce field named `nonce` created with `wp_create_nonce('mmhd_addon_nonce')`. The shared admin script already localizes this nonce as `mmhdAddon.nonce` (see Section 9). The capability required also varies per addon: `manage_options` for Zapier and Multi-Inbox save/test actions, `edit_posts` for WooCommerce, and `mmhd_access_dashboard` elsewhere.

8.6 Gating on the shared license

Because `MMHD_Addon_Manager::load_active_addons()` only requires an addon's main file when `mmhd_addon_has_valid_license()` is true, a custom addon automatically inherits the same license gate as the bundled addons — no extra code is required inside the addon itself.

9. AJAX API Reference

All AJAX endpoints are served through WordPress's admin-ajax.php (action names below are the values passed as action=... in the POST body). Shared admin actions verify a nonce created with wp_create_nonce('mmhd_addon_nonce') and require the mmhd_access_dashboard capability via mmhd_addon_admin_guard(). The nonce and ajaxurl are localized into JS as the mmhdAddon object (see admin/mmhd-addon-admin.php).

9.1 Core addon-manager actions

Action	Purpose
mmhd_addon_activate	Activates a bundled addon by slug (requires a valid license).
mmhd_addon_deactivate	Deactivates a bundled addon by slug.
mmhd_addon_license_activate	Activates a license (email + key) against the licensing API.
mmhd_addon_license_deactivate	Releases the current site's license activation.
mmhd_addon_license_check	Checks license status, either for supplied email/key or the saved settings.

9.2 Per-addon AJAX actions

Discord

AJAX action
mmhd_discord_disconnect
mmhd_discord_fetch_channels
mmhd_discord_get_webhook
mmhd_discord_save_bot_token
mmhd_discord_save_channel
mmhd_discord_save_settings
mmhd_discord_save_webhook
mmhd_discord_send_test_message
mmhd_discord_test_connection
mmhd_discord_trigger_reminder_now

Filter Emails

AJAX action
mmhd_save_filter_emails_settings

Multi-Inbox

AJAX action
mmhd_fetch_emails
mmhd_multi_inbox_dashboard_tickets
mmhd_multi_inbox_get_inbox_names
mmhd_multi_inbox_save_settings
mmhd_multi_inbox_test_connection

Canned Replies (Saved Replies)

AJAX action
mmhd_saved_replies_create
mmhd_saved_replies_delete
mmhd_saved_replies_get_all
mmhd_saved_replies_update

Signature

AJAX action
mmhd_get_user_signature
mmhd_signature_get_settings
mmhd_signature_save_settings

Slack

AJAX action
mmhd_slack_disconnect
mmhd_slack_get_credentials
mmhd_slack_get_oauth_url
mmhd_slack_get_status
mmhd_slack_get_taxonomy_channels

AJAX action
mmhd_slack_get_workspace_data
mmhd_slack_save_credentials
mmhd_slack_save_settings
mmhd_slack_send_reply
mmhd_slack_send_ticket
mmhd_slack_test_connection
mmhd_slack_test_message
mmhd_slack_test_reminder

WhatsApp

AJAX action
mmhd_change_ticket_status
mmhd_whatsapp_disconnect
mmhd_whatsapp_get_bot_settings
mmhd_whatsapp_get_credentials
mmhd_whatsapp_get_history
mmhd_whatsapp_get_inbox
mmhd_whatsapp_get_status
mmhd_whatsapp_save_bot_settings
mmhd_whatsapp_save_credentials
mmhd_whatsapp_save_settings
mmhd_whatsapp_send_message
mmhd_whatsapp_send_test_message
mmhd_whatsapp_test_connection

Rating & Review

AJAX action
mmhd_rating_review_save_settings

WooCommerce

AJAX action
mmhd_woocommerce_get_customer_orders

Zapier

AJAX action
mmhd_zapier_clear_log
mmhd_zapier_delete_webhook
mmhd_zapier_generate_key
mmhd_zapier_save_webhook
mmhd_zapier_test_webhook

Note: The AI Bot addon also registers 7 wp_ajax_ actions in addition to its REST routes (see Section 10.2): mmhd_bot_prepare_queue, mmhd_bot_crawl_batch, mmhd_bot_crawl_progress, mmhd_save_ai_bot_key, mmhd_save_active_model, mmhd_save_widget_settings, mmhd_test_ai_connection.

10. REST API Reference

10.1 Metamint Helpdesk Addon core

Method	Route	Notes
GET	/wp-json/mmhd-addon/v1/status	Public. Returns activation status of the addon plugin itself.

10.2 AI Bot (mmhd-bot/v1)

Files: addons/ai-bot/includes/rest-api.php, includes/REST/chat-widget-api.php, includes/REST/data-source-api.php, includes/REST/mcp-api.php.

Route	Purpose
/post-types	List post types available as a knowledge-base source.
/api-key (GET/POST)	Read or save the active AI provider's API key.
/widget-config	Public endpoint serving the chat-widget's runtime configuration.
/chat-stream	Public streaming chat endpoint that powers the front-end AI Bot widget.
/fetch-posts-init, /fetch-posts-batch, /fetch-posts-finalize	Three-phase batched crawl/index of site content into the knowledge base (avoids timeouts on large sites).
/custom-knowledge-update, /custom-knowledge-fetch	Manage manually-entered custom knowledge-base text.
/delete-knowledge	Remove indexed knowledge entries.
/get-filestore	Retrieve generated knowledge-pack files.
/abilities-status	Report which "Abilities" (tool functions) are registered/enabled for the bot.
/save-mcp-config, /get-mcp-config	Manage the bot's Model Context Protocol (MCP) server configuration.
/confirm-refund (POST)	Confirms a WooCommerce refund initiated via the bot. Requires <code>is_user_logged_in</code> .
/mark-guide-seen (POST)	Marks the setup guide as seen for the current admin. Requires <code>mmhd_access_dashboard</code> .

10.3 Zapier (mmhd-zapier/v1)

All endpoints require an X-API-Key header (or an `api_key` query parameter), validated with `hash_equals()` against the site's generated Zapier API key (option `mmhd_zapier_api_key`).

Method	Route	Purpose
POST	/create-ticket	Create a support ticket (title, customer_email, description, customer_name, priority, category).
POST	/add-reply	Add a reply to a ticket (ticket_id, message, author_type, author_name, author_email).
POST	/update-ticket	Change status / add a note (id, status, note).
POST	/create-user	Create a WordPress user (email, name).
POST	/delete-user	Delete a WordPress user (user_id) — also logs the deletion for the Deleted User trigger.
POST	/delete-ticket	Delete a ticket (ticket_id) — logs the deletion for the Deleted Ticket trigger.
GET	/list-tickets	List/filter tickets (status, search, limit, offset, created_after, modified_after).
GET	/list-replies	List replies for a ticket.
GET	/list-categories	List ticket categories (used for Zapier dynamic dropdowns).
GET	/list-priorities	List ticket priorities (used for Zapier dynamic dropdowns).
GET	/list-users	List users created after a given time (backs the New User trigger).
GET	/list-deleted-tickets	Deleted-ticket log (backs the Deleted Ticket trigger).
GET	/list-deleted-users	Deleted-user log (backs the Deleted User trigger).

Zapier triggers are polling-based (checked roughly every 15 minutes on the free Zapier tier) and call the REST endpoints above.

A legacy "Webhooks by Zapier" push mode is also supported for backward compatibility (requires a paid Zapier plan); webhook URLs entered under Settings → Zapier are POSTed to directly by class-webhook-delivery.php whenever the trigger hooks in Section 7.1 fire.

10.3.1 Create Ticket

POST /create-ticket

Creates a new support ticket (post type mmhd_ticket, status open, origin Zapier). If no user exists for customer_email, a WordPress user is created automatically. Fires the mmhd_after_ticket_created hook.

Parameter	Type	Required	Description
title	string	Yes	Subject of the ticket
customer_email	string	Yes	Customer email address (must be a valid email)
description	text	No	Full description; falls back to title

customer_name	string	No	Customer display name (defaults to email)
priority	string	No	Priority name (e.g. low, medium, high, critical); created as a term if missing
category	string	No	Category name; applied only if the term already exists

Response 200:

```
{"success": true, "ticket_id": 123 }
```

Errors: missing_fields 400, invalid_email 400, user_creation_failed 500, creation_failed 500.

10.3.2 Add Reply

POST /add-reply

Adds a reply (comment type ticket_replies) to an existing ticket. If author_email matches a user, the reply is linked to that user; otherwise the ticket's customer email is used. Fires the mmhd_after_reply_added hook.

Parameter	Type	Required	Description
ticket_id	integer	Yes	ID of the ticket to reply to
message	text	Yes	Reply content
author_type	string	No	agent or customer (default: customer); any other value becomes customer
author_name	string	No	Display name of the author
author_email	string	No	Email of the author

Response 200

```
{"success": true, "comment_id": 456 }
```

Errors: missing_fields 400, not_found 404 (ticket missing or not a ticket), comment_failed 500.

10.3.3 Update Ticket

POST /update-ticket

Changes a ticket's status. The parameter is id (the Zapier action maps its ticket_id field to id in the request body). Always writes an activity comment recording the change, fires mmhd_after_ticket_status_change, and mmhd_after_ticket_closed when the new status is closed.

Parameter	Type	Required	Description
id	integer	Yes	Ticket ID
status	string	Yes	open, in-progress, or closed
note	text	No	Optional note; if omitted, a default change message is written as the activity comment

Response 200

```
{ "success": true, "ticket_id": 123, "comment_id": 456, "old_status": "open", "new_status": "closed" }
```

comment_id may be null if the comment could not be written. Errors: missing_fields 400, not_found 404, invalid_status 400.

10.3.4 Create User

POST /create-user

Creates a WordPress user with a random password. The user cannot log in with a known password; this is intended for ticket-linked users.

Parameter	Type	Required	Description
email	string	Yes	Email address (must be valid and unique)
name	string	No	Display name (defaults to email)
role	string	No	subscriber (default), author, editor, mmhd_support_staff, mmhd_support_manager, administrator

Responses 200

```
{ "success": true, "user_id": 789, "email": "john@example.com", "name": "John Doe" }
```

Errors: missing_fields 400, invalid_email 400, user_exists 409, invalid_role 400, user_creation_failed 500.

10.3.5 Delete User

POST /delete-user

Permanently deletes a WordPress user. Administrators cannot be deleted via the API. The deletion is recorded in the deletion log (feeds the Deleted User trigger).

Parameter	Type	Required	Description
user_id	integer	Yes	ID of the user to delete

Response 200:

```
{ "success": true, "user_id": 789 }
```

Errors: missing_fields 400, not_found 404, cannot_delete_admin 403, deletion_failed 500.

10.3.6 Delete Ticket

POST /delete-ticket

Permanently deletes a support ticket (bypasses the trash). The deletion is recorded in the deletion log (feeds the Deleted Ticket trigger).

Parameter	Type	Required	Description
ticket_id	integer	Yes	ID of the ticket to delete

Response 200:

```
{ "success": true, "ticket_id": 123 }
```

Errors: missing_fields 400, not_found 404, deletion_failed 500.

10.3.7 List Tickets

GET /list-tickets

Returns a paginated list of tickets. Powers the New Ticket (created_after) and Status Changed (modified_after) triggers.

Parameter	Type	Required	Description
status	string	No	Filter by status: open, in-progress, closed
search	string	No	Search keyword
limit	integer	No	Results per page — max 100, default 20
offset	integer	No	Pagination offset (default 0)
created_after	string	No	ISO 8601 timestamp — tickets created after this time

Parameter	Type	Required	Description
modified_after	string	No	ISO 8601 timestamp — tickets modified after this time

Response 200:

```
{ "tickets": [ { "id": 123, "title": "Support request", "description": "Need help with...", "status": "open", "priority": "High", "category": "Support", "customer_email": "customer@example.com", "created_at": "2026-07-25T12:00:00Z", "updated_at": "2026-07-25T12:00:00Z" } ], "total": 1 }
```

10.3.8 List Replies

GET /list-replies

Returns replies (comment type ticket_replies). Powers the New Reply trigger.

Parameter	Type	Required	Description
ticket_id	integer	No	Filter to a single ticket. Optional — if omitted, replies from all tickets are returned
limit	integer	No	Results per page — max 100, default 20
offset	integer	No	Pagination offset (default 0)

Response 200:

```
{ "replies": [ { "id": 456, "ticket_id": 123, "author": "John Doe", "author_email": "john@example.com", "author_type": "customer", "message": "Thanks for the help!", "created_at": "2026-07-25T12:05:00Z" } ] }
```

10.3.9 List Users

GET /list-users

Returns WordPress users ordered by registration date (newest first). Powers the New User trigger.

Parameter	Type	Required	Description
created_after	string	No	ISO 8601 timestamp — users registered after this time
limit	integer	No	Results per page — max 200, default 50
offset	integer	No	Pagination offset (default 0)

Response 200:

```
{ "users": [ { "id": 1, "email": "john@example.com", "name": "John Doe", "created_at":  
"2026-07-27T12:00:00Z" } ], "total": 1 }
```

10.3.10 List Deleted Tickets

GET /list-deleted-tickets

Returns tickets permanently deleted via the API or the delete-ticket flow, read from the deletion log (keeps the last 500 records; returns at most 100). Powers the Deleted Ticket trigger.

Parameter	Type	Required	Description
created_after	string	No	Timestamp — deletions after this time. See Notes & Edge Cases (11.10.5) about timestamp format.

Response 200:

```
{ "deleted_tickets": [ { "id": 123, "title": "Support request", "deleted_at": "2026-07-27T12:00:00Z" } ]  
}
```

10.3.11 List Deleted Users

GET /list-deleted-users

Returns users permanently deleted via the API or the delete-user flow, read from the deletion log. Powers the Deleted User trigger.

Parameter	Type	Required	Description
created_after	string	No	Timestamp — deletions after this time. See Notes & Edge Cases (11.10.5) about timestamp format.

Response 200:

```
{ "deleted_users": [ { "id": 1, "title": "John Doe (john@example.com)", "deleted_at":  
"2026-07-27T12:00:00Z" } ] }
```

10.3.12 List Categories

GET /list-categories

Returns all terms of the mmhd_ticket_category taxonomy. Used by the Zapier app's authentication test and the Category dynamic dropdown.

Response 200:

```
{ "categories": [ { "id": 1, "name": "Support" }, { "id": 2, "name": "Billing" } ] }
```

10.3.13 List Priorities

GET /list-priorities

Returns all terms of the mmhd_ticket_priority taxonomy. Used by the Zapier app's Priority dynamic dropdown.

Response 200:

```
{ "priorities": [ { "id": 1, "name": "Low" }, { "id": 2, "name": "Medium" }, { "id": 3, "name": "High" }, { "id": 4, "name": "Critical" } ] }
```

10.4 Error Responses

Errors use the standard WordPress REST WP_Error shape:

```
{ "code": "error_code", "message": "Human-readable error description", "data": { "status": 400 } }
```

Status	Code(s)	Meaning
200	—	Success
400	missing_fields, invalid_email, invalid_status, invalid_role	Missing or invalid parameters
401 / 403	rest_forbidden	Missing or invalid API key (permission_callback fails; 401 for anonymous callers)
403	cannot_delete_admin	Administrator users cannot be deleted via the API
404	not_found	Ticket or user not found
409	user_exists	A user with this email already exists
500	creation_failed, user_creation_failed, comment_failed, deletion_failed	Internal server error

10.5 — Rating & Review (mmhd-rating/v1)

File: addons/rating-review/includes/rating-review-rest.php. All four routes are registered in a single `rest_api_init` closure; none of them use the shared `mmhd_addon_admin_guard()` — the customer-facing pair is guarded by ownership checks, and the staff pair by capability.

Route	Methods	Permission	Purpose
/review-status	GET	<code>__return_true</code>	Feature-detection endpoint for the frontend widget. Returns whether the addon is enabled, the active system, its options, and — only for a user who may view the ticket — the ticket status, any existing review, and <code>can_review</code> .
/review	POST	<code>mmhd_rating_review_public_permission()</code> + <code>mmhd_rating_review_can_view_ticket()</code>	Stores the one review a customer may leave on a closed ticket and posts it into the thread.
/customer-rating	GET, POST	<code>current_user_can('mmhd_access_dashboard')</code>	Reads or saves the staff-side rating for the ticket's customer. Universal per customer (stored as user meta), not per ticket.

GET /review-status

Parameter	Type	Required	Description
ticket_id	integer	Yes	Ticket to evaluate; must be an existing <code>mmhd_ticket</code>

Response 200:

```
{ "success": true, "enabled": true, "system": "star", "options": [ ... ], "access": true, "ticket_id": 123, "ticket_status": "closed", "ticket_source": "chat", "existing": null, "can_review": true }
```

Note: when access is false the response is still HTTP 200 and simply omits `ticket_id`, `ticket_status`, `existing` and `can_review`. The endpoint is public on purpose so the widget can hide itself without leaking ticket data. Errors: `invalid_ticket` 404.

POST /review

Parameter	Type	Required	Description
ticket_id	integer	Yes	Ticket being reviewed
rating_key	string	Yes	One of the active system's keys (e.g. star_5, good, like) — sanitize_key()
comment	text	No	Optional free-text comment (sanitize_textarea_field())

Response 200:

```
{ "success": true, "message": "Thank you for your feedback!", "existing": { "score": 5, "key": "star_5",  
"label": "5 Stars", "emoji": "★★★★★", "comment": "...", "at": 1788000000, "by": 12,  
"assignee_id": 4 } }
```

Errors: forbidden 403, invalid_ticket 400, disabled 400, invalid_rating 400, not_closed 400, already_reviewed 400.

GET/POST /customer-rating

GET takes ticket_id and returns the stored rating (or null). POST additionally takes rating_key and optional comment, and writes an internal note via the core mmhd_add_internal_ticket_note() helper so the change is visible in the ticket's Notes tab. Errors: invalid_ticket 400 / 404, invalid_rating 400.

The heading 10.4 Error Responses above becomes 10.5 Error Responses. Its body text is unchanged.

11. Addon-by-Addon Reference

11.1 Slack Integration

Connects Metamint Helpdesk to a Slack workspace via OAuth so agents can receive new tickets, reply from Slack threads, and get reminders for unanswered tickets. Bundled folder: addons/slack/ (slack-addon.php, includes/slack-handlers.php, includes/slack-settings-ui.php, includes/slack-taxonomy-fields.php).

Files

```
slack-addon.php  
includes/slack-handlers.php  
includes/slack-settings-ui.php  
includes/slack-taxonomy-fields.php
```

Settings tab

Registered via `mmhd_settings_tabs` (key 'slack') and rendered via `mmhd_settings_content` → `includes/slack-settings-ui.php`.

Hooks used

- `save_post_mmhd_ticket` / `mmhd_after_reply_added` — push ticket/reply events into the connected Slack channel.
- `mmhd_slack_check_unanswered_tickets` / `mmhd_slack_delayed_reply_notification` / `mmhd_slack_sync_event` — custom cron-driven reminder hooks.
- Taxonomy hooks (`created_mmhd_ticket_category`, `edited_mmhd_ticket_category`, `mmhd_ticket_category_add_form_fields`, `mmhd_ticket_category_edit_form_fields`) — allow mapping a ticket category to a specific Slack channel.
- Uses Select2 for searchable channel/category dropdowns (loaded on the mmhd-dashboard admin screen and on `mmhd_ticket_category` taxonomy screens).
- Stores OAuth credentials, workspace metadata, channel map and user map in dedicated options: `mmhd_slack_credentials`, `mmhd_slack_settings`, `mmhd_slack_channels`, `mmhd_slack_users`, `mmhd_slack_workspace_info`, `mmhd_slack_connected`.
- Defaults to 'active' out of the box (see Section 3.3).

11.2 Discord Integration

Sends ticket/reply notifications to a Discord channel via a bot token + webhook, with optional forum-thread naming and unanswered-ticket reminders. Folder: `addons/discord/`.

Files

```
discord-addon.php
includes/discord-handlers.php
includes/discord-settings-ui.php
includes/discord-category-ui.php
```

Settings tab

Registered under the 'discord' tab.

Hooks used

- `save_post_mmhd_ticket / mmhd_after_reply_added` — trigger webhook delivery.
- `mmhd_discord_check_unanswered_tickets`, `mmhd_discord_delayed_reply_notification`, `mmhd_discord_delayed_ticket_notification` — custom cron-driven reminders.

Implementation notes

- Credentials/state stored across `mmhd_discord_bot_token`, `mmhd_discord_application_id`, `mmhd_discord_guild_id`, `mmhd_discord_channel_id`, `mmhd_discord_webhook_url/id/token`, `mmhd_discord_thread_name`, `mmhd_discord_settings`.
- Ships a category UI (includes/discord-category-ui.php) so a category can be mapped to a dedicated Discord channel/thread.

11.3 Signature

Lets each WordPress user (agent) maintain a rich-text signature (WYSIWYG via `wp_editor`) that can optionally be appended to ticket replies. This is the simplest addon in the bundle and the recommended template to study before building a new one.

Files

```
signature-addon.php
includes/signature-handlers.php
```

Hooks used

- `show_user_profile / edit_user_profile` — render the signature WYSIWYG field on the user-profile screen.
- `personal_options_update / edit_user_profile_update` — persist the signature to user meta `mmhd_user_signature` (sanitized with `wp_kses_post`).
- `mmhd_after_reply_editor` — renders an 'Include signature in reply' checkbox directly under the ticket reply editor, but only when the addon's status option is 'active'.

Implementation notes

- `mmhd_addon_signature_deactivate()` clears the `mmhd_signature_cache` transient on deactivation — settings themselves are preserved until uninstall.
- Good reference for the minimal required shape of an addon: `ABSPATH` guard, `{slug}_init()`, optional `{slug}_deactivate()`, direct call to `{slug}_init()` at the bottom of the file.

11.4 Canned Replies (Saved Replies)

CRUD management of reusable reply templates agents can insert into the ticket editor. Folder: addons/saved-replies/ (addon.json slug is saved_replies, folder is saved-replies).

Files

```
saved-replies-addon.php
includes/saved-replies-handlers.php
```

Hooks used

- Registers a canned_reply custom post type, hooks mmhd_before_reply_editor to render its dropdown in the reply editor, and adds an admin submenu — in addition to integrating through its four AJAX endpoints and dashboard-side JS.

Implementation notes

- Templates are persisted in the mmhd_saved_replies option as a serialized list.

11.5 AI Bot

The most complex bundled addon: an on-site AI chat widget backed by a retrieval-augmented knowledge base (site content + custom text), a set of callable "Abilities" (structured tool functions the model can invoke), and optional MCP (Model Context Protocol) connectivity for external tool servers.

Files

```
ai-bot-addon.php
includes/rest-api.php, includes/crawl.php, includes/utils.php
includes/REST/chat-widget-api.php, data-source-api.php, mcp-api.php
includes/Abilities/AbilityRegistrar.php, FaqAbilities.php,
TicketAbilities.php, WordPressAbilities.php, WooCommerceAbilities.php
includes/Knowledge/KnowledgeIndexer.php
includes/Functions/LocalFunctions.php
assets/ + build/ (compiled React settings UI mounted into
#mmhd-ai-bot-settings-root)
```

Settings tab

Registered under 'ai_bot'; unlike the other addons, the settings tab body is not server-rendered PHP — it is an empty mount div (#mmhd-ai-bot-settings-root) hydrated by a bundled React app (assets under build/).

Hooks used

- Uses the WordPress core Abilities API (wp_abilities_api_init, wp_abilities_api_categories_init, wp_register_ability_args, wp_register_ability_category_args, wp_before_execute_ability, wp_after_execute_ability, abilities_api_register_core_abilities) to register callable tool functions such as FAQ lookup, ticket lookup, WordPress content search, and WooCommerce order lookup.

Implementation notes

- Namespace: Metamint Helpdesk\AiBot\... (e.g. Metamint Helpdesk\AiBot\Abilities\AbilityRegistrar). AbilityRegistrar::init() is called once at file-load time, before the usual mmhd_addon_ai_bot_init() runs on plugins_loaded, so abilities are available even before the settings screen is opened.
- Owns a dedicated custom database table (see Section 12.1) for FULLTEXT knowledge search, replacing an older wp_options-blob approach.
- Site content is indexed in three phases (fetch-posts-init / -batch / -finalize) specifically to avoid PHP execution timeouts when crawling large sites.
- Settings/options include mmhd_bot_active_model, mmhd_bot_chatgpt_api_key, mmhd_bot_gemini_api_key, mmhd_bot_primary_bot, mmhd_bot_widget_config, mmhd_bot_mcp_config, mmhd_bot_abilities_status, mmhd_bot_crawl_queue / mmhd_bot_crawl_progress.

11.6 Filter Emails

Lets an admin configure blacklist/wildcard rules that block specific senders or domains from auto-creating tickets when Metamint Helpdesk polls an email inbox.

Files

```
filter-emails-addon.php
includes/filter-emails-handlers.php
includes/filter-emails-settings-ui.php
includes/filter-emails-utils.php
```

Settings tab

Registered under 'filter-emails'.

Hooks used

- `mmhd_before_create_ticket_from_email` — the primary filter this addon hooks into; returning false/short-circuiting here blocks ticket creation for a matched sender.

Implementation notes

- Rules, activity log and running statistics are stored in `mmhd_filter_emails_settings`, `mmhd_filter_emails_log`, and `mmhd_filter_emails_statistics`.

11.7 WooCommerce

Surfaces a customer's recent WooCommerce order history directly inside the ticket/conversation detail view, so agents have order context without leaving Metamint Helpdesk.

Files

```
woocommerce-addon.php
includes/woocommerce-handlers.php
```

Hooks used

- No settings-tab or core notification hooks — purely a read-only data lookup surfaced via a single AJAX action, `mmhd_woocommerce_get_customer_orders`, keyed off the customer's email on the conversation.

Implementation notes

- Addon icon is the only bundled addon that points at an image asset (registry/Addons-grid icon is Font Awesome class ``fas fa-cart-shopping``) rather than a Font Awesome class.
- WooCommerce's `addon.json` declares the image icon (`assets/images/woocommerce-logo.svg`), but the registry / Addons-grid icon is the Font Awesome class `fas fa-cart-shopping`.

11.8 Multi-Inbox

Files

```
multi-inbox-addon.php
includes/multi-inbox-handlers.php
includes/multi-inbox-settings-ui.php
Settings tab
```

Registered under 'imap'.

Hooks used

- `mmhd_email_fetch_cron` — hooks the shared email-fetch cron to poll each configured inbox.
- `mmhd_before_create_ticket_from_email` — used to tag the resulting ticket with its source inbox name.

Implementation notes

- Persists connection settings and per-inbox last-fetched timestamps in `mmhd_multi_inbox_settings`, `mmhd_multi_inbox_last_dates`, `mmhd_last_fetched_date`, and (legacy/shared) `mmhd_imap_settings`.

11.9 WhatsApp Integration

Connects Metamint Helpdesk to the WhatsApp Business API (Cloud API) so customer conversations can be received, replied to, and optionally handled by an automated bot flow, directly from the Metamint Helpdesk dashboard.

Files

```
whatsapp-addon.php
includes/whatsapp-handlers.php
includes/whatsapp-settings-ui.php
includes/whatsapp-reply-hook.php
```

Settings tab

Registered under 'whatsapp'.

Hooks used

- `mmhd_whatsapp_new_conversation` / `mmhd_whatsapp_new_message` — custom actions fired when inbound WhatsApp traffic arrives, used to create/update Metamint Helpdesk tickets/conversations.

Implementation notes

- `includes/whatsapp-reply-hook.php` is required unconditionally on `plugins_loaded` priority 5 (before the license gate is even evaluated) — see Section 2.3 — because its purpose is to register `mmhd_whatsapp_send_reply_to_customer` on `mmhd_after_reply_added` for AJAX/REST reply

flows; the Meta webhook routes themselves live in the license-gated handlers. It also writes a debug log to WP_CONTENT_DIR/mmhd-whatsapp-reply.log on every request.

- Credentials, phone metadata, verify token and bot settings are stored in mmhd_whatsapp_credentials, mmhd_whatsapp_phone_info, mmhd_whatsapp_verify_token, mmhd_whatsapp_settings, mmhd_whatsapp_bot_settings, mmhd_whatsapp_connected.
- Registers REST webhook routes GET /webhook and POST /webhook (for Meta webhook verification and callback delivery), in the license-gated handlers.

11.10 Zapier Integration

Provides both webhook triggers (ticket.created, ticket.reply_added, ticket.status_changed, ticket.closed) and REST API actions (Create Ticket, Add Reply, Update Ticket, Create/Delete User, Delete Ticket) so Metamint Helpdesk can be wired into 8000+ apps via Zapier; New User, Deleted Ticket, and Deleted User are polling-only triggers, not webhook-based. See Section 10.3 for the full endpoint table and zapier-integration.md (bundled inside the addon folder) for the complete setup walkthrough.

Files

```
zapier-addon.php
includes/class-auth.php          # API key generation + validation
                                  (hash_equals)
includes/class-webhook-delivery.php # HTTP POST + 3 retries + delivery
                                  log
includes/class-trigger-handlers.php # listens to core do_action hooks,
                                  dispatches webhooks
includes/class-action-endpoints.php # REST + AJAX endpoints
includes/class-settings-ui.php    # Settings tab renderer
zapier-integration.md            # full developer + end-user guide,
                                  bundled in-repo
```

Settings tab

Registered under 'zapier'.

Hooks used

- mmhd_after_ticket_created, mmhd_after_reply_added, mmhd_after_ticket_status_change, mmhd_after_ticket_closed — all four are listened to by class-trigger-handlers.php to fire outbound webhooks (legacy method) and to feed the polling REST endpoints (primary method).

Implementation notes

- API key stored in `mmhd_zapier_api_key`; delivery log (last 100 entries) in `mmhd_zapier_delivery_log`; deletion log (backing the two 'Deleted' triggers) in `mmhd_zapier_deletions`; saved legacy webhook URLs in `mmhd_zapier_webhooks`.
- Defaults to 'active' out of the box (see Section 3.3).

Setup & Credentials

1. Install and activate Metamint Helpdesk (core) and Metamint Helpdesk Addon on your WordPress site.
2. Activate the Zapier Integration addon under Metamint Helpdesk → Addons.
3. Go to Metamint Helpdesk → Settings → Zapier. Your API key (auto-generated on first visit) and Site URL are shown.
4. In Zapier, search for "Metamint Helpdesk", choose Connect, and enter the Site URL and API Key. Zapier verifies the connection automatically.

Item	Where to find it
API Key	WordPress Admin → Metamint Helpdesk → Settings → Zapier (read-only field; use Regenerate to rotate — this invalidates the old key immediately)
Site URL	Same settings page (displayed below the API key)

Every REST request requires a valid API key.

- Preferred: `X-API-Key: rbz_your_api_key_here` header.
- Fallback: the query parameter `?api_key=YOUR_KEY` is also accepted.
- Keys are validated with a constant-time comparison (`hash_equals()`).

Zapier app connection rules (v1.0.0):

- HTTPS only. The Site URL must start with `https://`. Plain `http://` is rejected — all API traffic must use HTTPS.
- Enter only the site root: no paths, no query strings, no trailing slash.
- Values containing `/wp-admin` or `/wp-json` are rejected with clear guidance.

Example request:

```
curl -X POST "https://yoursite.com/index.php?rest_route=/mmhd-zapier/v1/create-ticket" -H
"X-API-Key: rbz_your_api_key_here" -H "Content-Type: application/json" -d
'{"title":"Hello","customer_email":"customer@example.com"}'
```

Zapier App Integration Details

The official Metamint Helpdesk Zapier app (v1.0.0, zapier-platform-core 19.1.0) works on all Zapier plans including free. It connects to your site, polls for new data, and exposes the following events.

Triggers (Metamint Helpdesk → Zapier, polling):

Zapier Trigger

Zapier Trigger	Endpoint polled	Polling parameters
New Ticket	GET list-tickets	limit=20, offset (per page), created_after = last poll
New Reply	GET list-replies	limit=20, offset, created_after = last poll
Status Changed	GET list-tickets	modified_after = last poll
New User	GET list-users	limit=50, offset (per page), created_after = last poll
Deleted Ticket	GET list-deleted-tickets	created_after = last poll
Deleted User	GET list-deleted-users	created_after = last poll

Two hidden dynamic-dropdown triggers — List Categories and List Priorities — power the category/priority selectors inside the Create Ticket action.

Actions (Zapier → Metamint Helpdesk):

Zapier Action	Request sent	Required fields	Optional fields
Create Ticket	POST create-ticket	title, customer_email	description, customer_name, priority (dropdown), category (dropdown)
Add Reply	POST add-reply	ticket_id, message	author_type, author_name, author_email
Update Ticket	POST update-ticket (sends id in the body, mapped from ticket_id)	ticket_id, status	note
Create User	POST create-user	email	name, role
Delete User	POST delete-user	user_id	—
Delete Ticket	POST delete-ticket	ticket_id	—

- All IDs are sent as JSON body parameters — never interpolated into the URL path.

- Every request carries X-API-Key (added by beforeRequest middleware) and Content-Type: application/json for POSTs.
- beforeRequest rewrites /wp-json/ → /index.php?rest_route=/ and strips any trailing slash from site_url, so a user-entered trailing slash can never produce a //index.php URL.
- Timestamps for polling (created_after, modified_after) are sent as ISO 8601 (lastPoll.toISOString()).

11.11 — Rating & Review

Two independent feedback loops on the same folder: customers rate the support they received on a closed conversation, and staff rate the customer. Both are keyed off ticket state ('_mmhd_ticket_status') rather than off a transport, so a conversation that arrived by chat, email, or WhatsApp behaves identically. Bundled folder: addons/rating-review/.

Files

```
rating-review-addon.php

includes/rating-review-functions.php      # settings, scoring, assignee
resolution, review storage, access checks

includes/rating-review-rest.php           # the four mmhd-rating/v1
routes

includes/rating-review-mail.php           # review CTA on the "ticket
closed" email

includes/rating-review-admin.php          # Users-list column, user
profile section, sidebar card markup

includes/rating-review-settings.php       # settings tab UI, save
handler, asset localization

assets/js/rating-review-widget.js         # frontend injection into the
chat widget

assets/js/rating-review-admin.js          # users column, profile log,
"Rate this customer" sidebar card

assets/css/rating-review.css

docs/user-guide.md, developer-guide.md, api-reference.md, support.md  #
full addon documentation
```

Settings tab

Registered under 'rating-review'. The tab renders server-side PHP; the tab slug uses a hyphen while the addon key in the registry is the underscore form `rating_review`.

Hooks used

- `mmhd_settings_tabs / mmhd_settings_content` — the settings tab.
- `mmhd_after_ticket_closed` — records the just-closed ticket id in `$GLOBALS['mmhd_rr_last_closed_ticket']` so the outgoing close email can be matched to it.
- `wp_mail` (priority 20) — appends the review CTA. The addon matches on the exact core subject 'Your ticket has been closed' and injects before `</body>`, so no core edit is required; the global is cleared one-shot so a second email in the same request can never inherit the link.
- `manage_users_columns / manage_users_custom_column` — the "Customer Reviews" column on Users → All Users.
- `show_user_profile / edit_user_profile` — the "Metamint Helpdesk" section on the user profile.
- Core functions consumed (not hooks): `mmhd_build_ticket_deeplink()`, `mmhd_get_ticket_customer_user_id()`, `mmhd_add_internal_ticket_note()`, `mmhd_get_requested_ticket_id_from_query()`, `mmhd_validate_token()`. Every call is guarded by `function_exists()` so the addon degrades rather than fatals against an older core.

Implementation notes

- Scoring is defined once in `mmhd_rating_review_get_system_options()` as `key/label/emoji/score`, and always normalised to a 1–5 scale. `mmhd_rating_review_meta_for_key()` recovers a stored key's system by scanning all systems, so reviews given under a previous system keep their label and score and the reports never shift meaning.
- Assignee attribution is resolved in order: `ticket_assignee_id` post meta → the `mmhd_default_agent` (or legacy `mmhd_default_staff`) option, which may be an ID, a login, or a display name → the first administrator. `mmhd_review_assignee_id` is written at review time so later reassignment does not rewrite history.
- Reviews are stored as post meta, and a summary comment (type `ticket_replies`, with comment meta `comment_source=admin`, `author_type=agent`, `mmhd_review=1`) is inserted into the thread credited to the assignee. The customer is recorded in `mmhd_review_by` via `mmhd_get_ticket_customer_user_id()`, falling back to the current user.
- The staff-side customer rating is universal per customer: it is written to user meta (`mmhd_customer_rating_*`) whenever a customer user can be resolved from `post_author` or `_mmhd_customer_email`, and only falls back to ticket meta for legacy tickets. A note is written to the ticket's internal Notes tab on every save.
- Aggregate agent scores are computed by a single joined postmeta query (`mmhd_rating_review_assignee_stats()`) and memoised in a static, so the Users list renders

in one query regardless of row count; the per-agent log is a separate prepared query (mmhd_rating_review_assignee Rated_tickets()).

- There is no build step and no custom table. The frontend widget is injected by wrapping window.fetch to learn the open ticket id from the core widget's own REST calls, then painting a panel below the message list. The single-ticket sidebar card is injected by JS for the same reason — the core dashboard sidebar exposes no hook.
- Assets are enqueued unconditionally on ?page=mmhd-dashboard (settings) and on wp_enqueue_scripts when enabled, so the enable toggle renders correctly even while the addon is switched off.
- Non-destructive by design: mmhd_addon_rating_review_deactivate() is intentionally empty, and reviews/settings survive deactivation and reactivation.

12. Database & Storage Reference

12.1 Custom database table

wp_mmhd_knowledge_chunks (created/managed by ai-bot/includes/Knowledge/KnowledgeIndexer.php via dbDelta(), idempotent — safe to call on every activation):

```
CREATE TABLE {prefix}mmhd_knowledge_chunks (
  id          BIGINT UNSIGNED NOT NULL AUTO_INCREMENT,
  source_type VARCHAR(50)      NOT NULL,
  source_key  VARCHAR(255)     NOT NULL,
  url         VARCHAR(500)     NOT NULL DEFAULT '',
  chunk_text  MEDIUMTEXT      NOT NULL,
  updated_at  DATETIME         NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  PRIMARY KEY (id),
  INDEX idx_source_key (source_key(191)),
  INDEX idx_source_type (source_type),
  FULLTEXT KEY ft_chunk (chunk_text)
) ENGINE=InnoDB {charset_collate};
```

KnowledgeIndexer::table() returns the fully-prefixed table name; KnowledgeIndexer::drop_table() removes it (intended for an uninstall routine). This table backs the AI Bot's site__search_knowledge() FULLTEXT lookup and supersedes an earlier design that stored large blobs directly in wp_options.

12.2 WordPress options reference

Every addon persists its configuration through the standard WordPress Options API (get_option / update_option / delete_option). A second custom table, wp_mmhd_tool_audit_log, also exists in addition to the AI Bot knowledge table above (Section 12.1).

Option	Owner	Purpose
mmhd_addon_activated	core	Whether the addon plugin itself has completed activation.
mmhd_addon_{slug}_status	core	'active' 'inactive' status for each bundled addon, e.g. mmhd_addon_slack_status.
mmhd_addon_license_settings	licensing	Saved { email, key } license credentials.
mmhd_addon_license_state	licensing	Current license state (see Section 4.1).
mmhd_addon_license_meta	licensing	Cached { licenseId, orderId, expiresAt, checkedAt }.
mmhd_slack_credentials / _settings / _channels / _users / _workspace_info / _connected	Slack	OAuth tokens, channel/category map, cached workspace data, connection flag.
mmhd_discord_bot_token / _application_id / _guild_id / _channel_id / _webhook_url / _webhook_id / _webhook_token / _thread_name / _settings	Discord	Bot credentials, webhook target, thread naming, general settings.
mmhd_signature_content / _enabled / _position	Signature	(legacy/global) signature defaults — per-user signatures live in user meta mmhd_user_signature.
mmhd_saved_replies	Canned Replies	Serialized list of saved reply templates.
mmhd_bot_* (active_model, chatgpt_api_key, gemini_api_key, primary_bot, widget_config, mcp_config, abilities_status, custom_knowledge, wpack_content, crawl_queue, crawl_progress, indexed_posts, knowledge_table_version, title_name, profile_image, talk_to_human, guide_seen)	AI Bot	Provider keys/model choice, widget configuration, MCP config, ability toggles, knowledge-base source data and crawl state.
mmhd_filter_emails_settings / _log / _statistics	Filter Emails	Blacklist rules, activity log, running counters.
mmhd_imap_settings, mmhd_multi_inbox_settings / _last_dates, mmhd_last_fetched_date	Multi-Inbox	Per-inbox IMAP connection settings and fetch bookkeeping.

Option	Owner	Purpose
mmhd_whatsapp_credentials / _phone_info / _verify_token / _settings / _bot_settings / _connected	WhatsApp	Cloud API credentials, webhook verify token, bot flow configuration.
mmhd_zapier_api_key / _delivery_log / _deletions / _webhooks	Zapier	API key, outbound delivery log, delete-event log, saved legacy webhook URLs.
mmhd_io_licensing_secret	licensing	Actively read by the same-site/self-hosted auto-update path (Section 5.3) to look up the license key — not merely reserved.
mmhd_rating_review_settings	Rating & Review	{ enabled, review_system, prompt_title, comment_placeholder, submit_label, success_message, already_rated, email_link_enabled, email_link_text }.

Beyond options, Rating & Review is the only bundled addon that also persists to post meta, user meta and comments. Its keys are:

Key	Where	Purpose
mmhd_review_score	ticket post meta	Normalised 1–5 score; its presence is what mmhd_rating_review_exists() tests, so it doubles as the "already reviewed" flag
mmhd_review_rating_key	ticket post meta	Raw rating key as chosen (e.g. star_5, good, like) — sanitize_key()
mmhd_review_comment	ticket post meta	Optional customer comment
mmhd_review_at / mmhd_review_by	ticket post meta	Unix timestamp of the review and the WP user ID of the customer who left it

mmhd_review_assignee_id	ticket post meta	Agent credited at review time (resolved once, so reassignment does not rewrite history)
mmhd_review_comment_id	ticket post meta	The thread comment inserted for the review
mmhd_customer_rating_score / _key / _comment / _at / _by	user meta (ticket meta fallback)	Staff-side customer rating, shared across every conversation that customer has
comment_source / author_type / mmhd_review	comment meta	admin / agent / 1 — marks the thread comment as an addon-generated review

13. Security Guidelines

Follow these conventions — already used consistently across every bundled addon — when adding new code.

- Every PHP file starts with `if ( ! defined('ABSPATH') ) { exit; }` to prevent direct access outside of WordPress.
- Only the 5 core addon-manager/license AJAX actions (Section 9.1) call the shared `mmhd_addon_admin_guard()` with nonce `mmhd_addon_nonce`. Every per-addon handler instead verifies its own per-action nonce before touching `$_POST` (e.g. `mmhd_slack_credentials_action`, `mmhd_discord_save_bot_token`, `mmhd_signature_save_settings`, `mmhd_saved_replies_create`).
- Nonces are created per-action (the 5 core actions use `mmhd_addon_nonce` via `wp_create_nonce()/wp_verify_nonce()`; each addon's own handlers create and verify their own nonce). The capability required also varies per addon: `manage_options` for Zapier and Multi-Inbox save/test actions, `edit_posts` for WooCommerce, and `mmhd_access_dashboard` (a custom capability defined by Metamint Helpdesk core) elsewhere.
- All superglobal reads are wrapped in `isset()/sanitize_text_field()/sanitize_email()/wp_unslash()` as appropriate before use; output is escaped with `esc_html__()/esc_attr__()` or run through `wp_kses_post()` for rich content such as the Signature editor.
- External REST endpoints that accept writes (Zapier's `/create-ticket` etc.) are protected by an API key compared with `hash_equals()` (timing-safe comparison) rather than `==` or `===`.
- Feature gating (Section 4) means an invalid/expired license cannot be worked around client-side: `MMHD_Addon_Manager::load_active_addons()` never even `require_once()`s an addon's PHP file unless the server confirms the license is active.

14. Coding Standards & Tooling

14.1 Autoloading

The plugin uses the Jetpack Autoloader (vendor/autoload_packages.php) rather than the raw Composer autoload.php, which avoids autoloader collisions when multiple plugins on the same site bundle overlapping Composer dependencies.

14.2 Coding standard

phpcs.xml at the plugin root configures PHP_CodeSniffer with the WordPress Coding Standards (WPCS) ruleset (vendor/wp-coding-standards, vendor/phpcsstandards, vendor/squizlabs, vendor/dealerdirect for the Composer PHPCS plugin integration). Run the sniffer from the plugin root:

```
vendor/bin/phpcs --standard=phpcs.xml .
```

14.3 Function naming convention

Pattern	Example	Purpose
mmhd_addon_{slug}_init()	mmhd_addon_slack_init()	Called automatically when the addon is loaded.
mmhd_addon_{slug}_deactivate()	mmhd_addon_signature_deactivate()	Called automatically when the addon is deactivated, if it exists.
mmhd_addon_{slug}_{action}()	mmhd_addon_discord_save_settings()	AJAX handler / feature function, hung off wp_ajax_mmhd_{slug}_{action}.
mmhd_{slug}_{noun}	mmhd_slack_settings	WordPress option name used for persistence.

14.4 Text domain

All translatable strings across the addon plugin and every bundled addon use the single text domain metamint-helpdesk-addon, loaded from /languages via load_plugin_textdomain() on init.

Appendix A: addon.json Field Reference

Field	Required	Notes
slug	Yes	Underscore form matching the array key used in register_addons() and every mmhd_addon_{slug}_* function/option name.
name	Yes	Display name; wrapped in __() with the metamint-helpdesk-addon field.
description	Yes	One-line description shown in the Addons grid.
version	Yes	Semantic version of the addon module itself.
icon	Yes	Font Awesome class or a path to an image asset relative to the addon folder.
icon_bg / icon_color	Yes	Tailwind-style color tokens for the icon badge background/foreground.
main_file	Yes	Bootstrap PHP filename, relative to the addon folder.
settings_tab	No	Tab slug if the addon registers a Settings tab; omit if it has none.
text_domain	Yes	Always metamint-helpdesk-addon for bundled addons.

Appendix B: Quick Reference Cheat Sheet

I want to...	Do this
React to a new ticket	add_action('mmhd_after_ticket_created', \$callback)
React to a new reply	add_action('mmhd_after_reply_added', \$callback)
React to a status change	add_action('mmhd_after_ticket_status_change', \$callback)
Add a Settings tab	add_filter('mmhd_settings_tabs', ...) + add_action('mmhd_settings_content', ...)
Block ticket creation from an inbound email	add_filter('mmhd_before_create_ticket_from_email', ...)
Add UI right under the reply editor	add_action('mmhd_after_reply_editor', ...)
Check if the addon license is valid	mmhd_addon_has_valid_license()
Check if mmhd core is active	mmhd_addon_is_mmhd_active()
Get the Addon Manager	mmhd_addon_manager()
Get the License Service	mmhd_addon_license_service()
Verify an admin AJAX request	mmhd_addon_admin_guard()
Check whether a closed ticket is reviewable	GET /mmhd-rating/v1/review-status?ticket_id={id} (or mmhd_rating_review_is_enabled())
Submit a customer review	POST/mmhd-rating/v1/review (or mmhd_rating_review_save_review())

I want to...	Do this
Read or save a ticket customer's rating	GET/POST/mmhd-rating/v1/customer-rating(or mmhd_rating_review_get_customer_rating() / mmhd_rating_review_save_customer_rating())
Get an agent's average customer rating	mmhd_rating_review_assignee_stats()[\$agent_id] — 'count' and 'avg'
Get the Rating & Review settings	mmhd_rating_review_get_settings()