Project Tracker - Development Document v0.1

1. Project Overview

- **Project Name:** Project Tracker
- **Purpose:** To provide a WordPress-based system for city staff to track the progress, status, files, and details of city projects.
- **Target Users:** City staff (e.g., Public Works Director Phillip Rawlings).
- **Foundation:** Based on Property Book plugin (v1.0 Stable) and Code Ruleset 1. *Note: The Property Book Development Document () should be considered a companion to this document, as it establishes the core Code Ruleset 1 and development patterns (e.g., for modals, AJAX, DataTables, security) that this Project Tracker plugin aims to follow for consistency.*
- **Key Goal:** Maintain consistency with Property Book structure and UI/UX where applicable.

2. Code Ruleset 1 (WordPress Plugin Development)

(Adapted for Project Tracker)

2.1. Project Setup & Basics:

- Use full table names as provided by the user; do not assume or add database prefixes unless instructed (e.g., `PrTr_projects`).
- Assume nothing exists (functions, variables, data, files) unless explicitly stated in the project documentation or current code baseline.
- If creating new elements (files, functions, database tables/columns) is necessary to fulfill a prompt, inform the user before providing the implementation code.
- Be aware that the DataTables JS library is assumed to be available and automatically applied to HTML tables generated by the plugin, unless specified otherwise.
- Ensure no plugin code interacts with or disrupts the WordPress admin area (`/wp-admin`), unless the task specifically involves admin-area functionality.
- User access control to the WordPress page displaying the plugin output is considered pre-handled, unless managing user permissions or roles is explicitly part of the project requirements.

2.2. Security:

- Adhere strictly to WordPress security best practices. This includes, but is not limited to:

    - **Input Sanitization:** Sanitize all data received from users or external sources before using it in database queries, function calls, or displaying it. Use appropriate WordPress sanitization functions (e.g., `sanitize_text_field`, `sanitize_textarea_field`, `intval`, `absint`).
    - **Nonces:** Use WordPress nonces (e.g., `wp_create_nonce`, `wp_verify_nonce`) for all form submissions and AJAX actions that perform database changes or sensitive operations to prevent CSRF attacks.
    - **Output Escaping:** Escape all data before outputting it to the browser to prevent XSS attacks. Use appropriate WordPress escaping functions based on the context (e.g., `esc_html`, `esc_attr`, `esc_js`, `esc_url`).
    - **Database Queries:** Use `$wpdb->prepare` for all database queries involving variables to prevent SQL injection vulnerabilities.
    - **Permissions Checks:** Verify user **roles** appropriately before allowing actions that modify data or expose sensitive information. Use roles for authorization checks (e.g., `in_array()`). Do not use WordPress capabilities unless explicitly instructed.
    - If a user's prompt requests an action that introduces a security concern or deviates from best practices, inform the user immediately before proceeding.
    - **Prevent Direct File Access:** Implement `defined('ABSPATH') || exit;` at the very top of all PHP files within the plugin to prevent them from being accessed directly via URL.

2.3. Code Formatting & Comments:

- **Organization:** Organize code logically by language (PHP, JS, CSS) and purpose. Group related functions or styles together.
- **Sectioning:** Use comment blocks to clearly delineate logical sections of code.
    - **Major Sections:**
        - Represent distinct functional areas (e.g., "Project Data Display," "Add/Edit Project Modal," "File Upload Handling").
        - Use for significant blocks implementing a larger process/feature.
        - Must have a header comment block explaining purpose and a footer comment block indicating the expected return value (or "No explicit return value").
        - Major sections should not be nested within other major sections.
        - *Format:*

PHP

```
Unset

/*
```

```
START SECTION --- [Major Section Name] ---
[Optional: Short description of the section's purpose and
functionality.]
*/

// ... code for the major section ...

/*
[Expected return value description, or "No explicit return
value."]
END SECTION --- [Major Section Name] ---
*/
```

- **Subsections:**
  - Represent smaller, specific units within a major section.
  - Use for: Individual functions; Logical blocks within a function (e.g., "Data Retrieval," "Input Validation").
  - Can be nested if it improves clarity.
  - Require simpler start/end comment markers.
  - *Format (e.g., Functions):*

PHP

Unset

```
/* START [subsection_name] */
function subsection_name() {
    // ... function code ...
}
/* END [subsection_name] */
```

- **Comments within Functions:**
  - Use liberally (`//` or `/* ... */`) to explain logic ("why"), purpose of variables, and execution flow.
  - Focus on "why," not just "what".
  - Keep concise and up-to-date.
  - Use to delineate distinct steps.

2.4. Code Provision Workflow:

- **New Features/Files:** Provide complete section(s) required, including standard header/footer comment blocks.
- **Modifications:**
    - Provide the smallest complete, logical block necessary for the change (e.g., a function, CSS rule block, loop, subsection).
    - **Context is Key:** Always provide sufficient context (e.g., surrounding lines) for placement.
    - Clearly identify the block to be replaced using existing unique comment markers or unambiguous surrounding code.
    - **Targeted Replacement:** Aim for direct copy-paste replacement where possible.
- **Placeholders:** Avoid placeholders like `// [Your code here]`. Complete steps sequentially or use subsections [cite: 867, 868, 876-878].

2.5. HTML/JS Output Structure:

- Prefer closing/reopening PHP tags (`?> ... <?php`) for large HTML/JS blocks instead of `echo`.
- **Crucially:** All dynamic PHP variables within HTML/JS blocks must be properly escaped for the output context (`esc_attr()`, `esc_html()`, `esc_js()`, etc.).
- Use correct comment syntax for the containing language (e.g., `<?php /* ... */ ?>` within PHP tags outputting HTML/JS).
- Match indentation of existing code when providing modifications.

2.6. File Structure (Project Specific - Project Tracker):

- Assume all PHP, CSS, and JS files reside in the main plugin directory (`project-tracker` or user-specified name) unless otherwise noted.
- **Core Files:**
    - `PrTr_init.php`: Main plugin file. Responsible for including other PHP files, enqueuing CSS/JS, defining shortcodes, registering AJAX actions, and handling POST submissions.
    - `PrTr_project.js`: All frontend JavaScript (DataTables init, modal interactions, filtering, lightbox, print trigger).
    - `PrTr_style.css`: All CSS styles (base, modals, filters, print).
    - *(Optional `PrTr_functions.php`: Could be used for helper PHP functions if `PrTr_init.php` becomes too large, included by `PrTr_init.php`).*
- **Upload Directory:**
    - `wp-content/uploads/PrTr_files/` (This subdirectory within the standard WordPress uploads path must be created and have correct write permissions).

2.7. Project Workflow & Information Gathering:

- **Ruleset Confirmation:** Confirm understanding of this ruleset when starting work or requested.
- **Initial Information:** Request (if not provided): Plugin name, directory name, DB schema (if changed), current status, overall goals.
- **Existing Projects:** Ask if this ruleset was used previously. If not, assess/refactor existing code first.
- **Clarification:** Ask follow-up questions to ensure clear understanding before coding.
- **Step-by-Step:** Address tasks one logical step/modification at a time. Provide code, await user confirmation before proceeding.

2.8. General Principles:

- **Prioritize Quality:** Correct, secure, maintainable code over speed.
- **Mimic Working Code:** Follow logic, structure, style of similar working examples (e.g., from Property Book) where possible. Explain necessary deviations.
- **Backup:** Remind user periodically to back up files before applying changes.
- **Debugging:** Use browser console (`console.log`) for debugging messages. Server error logs are unavailable.

2.9. Project Expansion:

- Different rulesets may apply for major expansions.
- If expansion occurs: Understand scope, ask about changes (structure, DB, dependencies), and ask if ruleset will be modified/replaced.

3. Database Design

- **Database Prefix:** `PrTr_`
- **New Tables:**
    - **`PrTr_projects`**:

```
Unset
CREATE TABLE `PrTr_projects` (
  `project_id` INT NOT NULL AUTO_INCREMENT,
  `display_id` VARCHAR(255) NULL DEFAULT NULL,
  `department_id` INT NOT NULL, -- FK to inv_departments.department_id
  `category_id` INT NOT NULL, -- FK to PrTr_project_categories.category_id
  `scheduled_start_date` DATE NULL DEFAULT NULL, -- Changed to NULLable based
on previous field analysis, confirm if needed
  `expected_completion_date` DATE NULL DEFAULT NULL, -- Changed to NULLable,
confirm if needed
  `project_completed_date` DATE NULL DEFAULT NULL,
```

```
  `client_department` INT NULL DEFAULT NULL, -- FK to
inv_departments.department_id
  `location` VARCHAR(255) NULL DEFAULT NULL, -- Changed to NULLable, confirm if
needed
  `status_id` INT NOT NULL, -- FK to PrTr_project_statuses.status_id
  `value` DECIMAL(10,2) NULL DEFAULT 0.00, -- Added default
  `expenses` DECIMAL(10,2) NULL DEFAULT 0.00, -- Added default
  `associated_purchase_orders` TEXT NULL DEFAULT NULL,
  `description` TEXT NULL DEFAULT NULL, -- Changed to NULLable, confirm if
needed
  `comments` TEXT NULL DEFAULT NULL,
  `public_release` TINYINT(1) NOT NULL DEFAULT 0, -- BOOLEAN equivalent
  `created_at` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `updated_at` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
  `updated_by` VARCHAR(60) NULL DEFAULT NULL, -- Assumes storing user_login

  PRIMARY KEY (`project_id`),

  INDEX `idx_department_id` (`department_id`),
  INDEX `idx_category_id` (`category_id`),
  INDEX `idx_status_id` (`status_id`),
  INDEX `idx_client_department` (`client_department`)

  -- Optional: Define Foreign Key Constraints explicitly if desired/needed
  -- CONSTRAINT `fk_project_dept` FOREIGN KEY (`department_id`) REFERENCES
`inv_departments` (`department_id`),
  -- CONSTRAINT `fk_project_cat` FOREIGN KEY (`category_id`) REFERENCES
`PrTr_project_categories` (`category_id`),
  -- CONSTRAINT `fk_project_status` FOREIGN KEY (`status_id`) REFERENCES
`PrTr_project_statuses` (`status_id`),
  -- CONSTRAINT `fk_project_client_dept` FOREIGN KEY (`client_department`)
REFERENCES `inv_departments` (`department_id`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

- **PrTr_project_categories**:

```
Unset
CREATE TABLE `PrTr_project_categories` (
  `category_id` INT NOT NULL AUTO_INCREMENT,
```

```
  `category_name` VARCHAR(255) NOT NULL,

  PRIMARY KEY (`category_id`),
  UNIQUE KEY `idx_category_name` (`category_name`) -- Ensure category names are
unique

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

- **PrTr_project_statuses**:

```
Unset
CREATE TABLE `PrTr_project_statuses` (
  `status_id` INT NOT NULL AUTO_INCREMENT,
  `status_name` VARCHAR(100) NOT NULL,
  `status_order` INT NOT NULL DEFAULT 0, -- For custom sorting

  PRIMARY KEY (`status_id`),
  UNIQUE KEY `idx_status_name` (`status_name`), -- Ensure status names are
unique
  INDEX `idx_status_order` (`status_order`) -- Index for sorting

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- Optional: Pre-populate with defined statuses after table creation
-- INSERT INTO `PrTr_project_statuses` (`status_name`, `status_order`) VALUES
-- ('In Progress', 1),
-- ('Planning', 2),
-- ('On Hold', 3),
-- ('Completed', 4),
-- ('Cancelled', 5);
```

- **PrTr_project_files**:

```
Unset
CREATE TABLE `PrTr_project_files` (
  `file_id` INT NOT NULL AUTO_INCREMENT,
  `project_id` INT NOT NULL, -- FK to PrTr_projects.project_id
  `file_name` VARCHAR(255) NOT NULL, -- Original filename
```

```
  `file_path` VARCHAR(512) NOT NULL, -- Path relative to WP uploads dir
  `file_type` VARCHAR(100) NOT NULL, -- MIME type
  `uploaded_at` TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `uploaded_by` VARCHAR(60) NULL DEFAULT NULL, -- Assumes storing user_login

  PRIMARY KEY (`file_id`),
  INDEX `idx_project_id` (`project_id`) -- Index for joining with projects

  -- Optional: Define Foreign Key Constraint
  -- CONSTRAINT `fk_file_project` FOREIGN KEY (`project_id`) REFERENCES
`PrTr_projects` (`project_id`) ON DELETE CASCADE -- Consider CASCADE delete

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

- **Reused Tables:**
  - `inv_departments`: Used for `department_id` and `client_department` FKs. Need `department_id` and `department_name`.
  - *(WordPress `users` and `usermeta` tables implicitly used for getting user info and roles)*.

4. Plugin File Structure

- **Directory:** `project-tracker` (suggested name)
- **Files:**
  - `PrTr_init.php`: Main file (details in section 2.6).
  - `PrTr_project.js`: Frontend JavaScript (details in section 2.6).
  - `PrTr_style.css`: CSS styles (details in section 2.6).
  - `uploads/PrTr_files/`: Directory within `wp-content/uploads/` for file storage.

5. Core Functionality

- **Shortcode (`[project_tracker_list]`):**
  - Placed on a WordPress page to display the project tracker interface.
  - Outputs main table container (`#pt-projects-main-table`), filter container (`#pt-filters-container`), action buttons, and modals.
  - Handles displaying success/error feedback messages after POST submissions (similar to Property Book `pb-feedback-message` [cite: 75, 92, 124, 691-698]).

- **DataTables View (`#pt-projects-main-table`):**
    - Initialized by `PrTr_project.js`.
    - AJAX data source: `PrTr_get_projects_ajax_data` action handled in `PrTr_init.php`. Query must join relevant tables (`PrTr_projects`, `inv_departments`, `PrTr_project_categories`, `PrTr_project_statuses`) to get names for display. Query must filter based on user roles (see User Access Control).
    - Columns: Display relevant fields from `PrTr_projects` and joined tables. Hide internal IDs (`project_id`, `department_id`, `category_id`, `status_id`, `client_department` ID). Format dates, currency.
    - Default Sort: `status_order` (ASC), then `updated_at` (DESC).
    - Responsive column hiding using CSS classes (e.g., `pt-col-desktop-only`) applied via PHP/JS, similar to Property Book [cite: 109-111, 127-129, 714-720].
    - Actions Column:
        - "Details" button: Always visible for users with at least `view_projects` role on accessible rows. Opens `#pt-details-modal`.
        - "Edit" button: Visible only if user has `edit_projects` role AND the project's `department_id` matches one of the user's associated departments (OR user has `full_project_access` + `edit_projects`). Opens `#pt-project-modal` pre-filled.
- **In-Line Filtering (`#pt-filters-container`):**
    - HTML generated by PHP function called from shortcode (similar to Property Book `pb_render_filter_row_html`).
    - Displayed above the DataTables controls.
    - Controls:
        - Dropdown: Department (Filter by `department_id`. Options filtered based on user access).
        - Dropdown: Category (Filter by `category_id`).
        - Dropdown: Status (Filter by `status_id`).
        - Toggle/Dropdown: Public Release (Filter by `public_release` boolean - Yes/No/All).
        - Date Range: Two date inputs (Start Date, End Date). Filter rows where (`scheduled_start_date` <= End Date AND `expected_completion_date` >= Start Date) OR (`project_completed_date` BETWEEN Start Date AND End Date). *Refined logic needed here.*
        - Button: "Clear Filters".
    - JavaScript listeners in `PrTr_project.js` trigger DataTables `.search()` and `.draw()` on filter changes (similar to Property Book [cite: 210, 211, 224, 228, 230, 486-509]).

- **Add/Edit Modal (`#pt-project-modal`):**
    - Single modal used for both Add (empty form) and Edit (pre-filled form).
    - Triggered by "Add New Project" button (visible only for `edit_projects` role) and "Edit" button in table rows.
    - Form Fields: Correspond to `PrTr_projects` table columns, using appropriate input types (`text`, `date`, `number`, `textarea`, `select`, `checkbox` for `public_release`). Populate dropdowns (Department, Category, Status, Client Department) via PHP. Department dropdowns filtered by user access.
    - Layout: Use Fieldsets for logical grouping (e.g., "Project Details", "Dates", "Financials", "Files"). Adapt Property Book modal CSS [cite: 88, 117, 627-690].
    - File Upload Section:
        - HTML `input type="file"` (allow multiple, up to 5).
        - Display list of currently associated files (from `PrTr_project_files`).
        - Include a mechanism (e.g., button/icon next to each file) to delete existing files (triggers AJAX call to delete).
    - Data Submission: Standard HTML Form POST request (no AJAX). Handled by PHP function hooked to `init`.
    - POST Handler (PHP):
        - Verify nonce.
        - Verify user role (`edit_projects`) and permission to edit *this specific* project (department match or full access).
        - Sanitize and validate all incoming `$_POST` data.
        - Handle file uploads using `wp_handle_upload()`. Check count (< 5 total). Save file info to `PrTr_project_files`.
        - Handle file deletions requested via `$_POST`. Remove server file and DB record.
        - Perform `$wpdb->insert` (if no project ID) or `$wpdb->update` (if project ID exists) on `PrTr_projects`. Use `$wpdb->prepare`.
        - Update `updated_at` and `updated_by`.
        - Redirect back to the page with success/error query parameter (e.g., `?save_status=success`).
- **Details Modal (`#pt-details-modal`):**
    - Read-only view. Opened by "Details" button.
    - Populated via AJAX call (`PrTr_get_project_details`) triggered by button click. AJAX handler fetches project data and associated file list.
    - Layout similar to Add/Edit modal fieldsets, but using `<span>` or `<div>` for data display.
    - Image Thumbnails Section: Display uploaded images scaled down via CSS. Each thumbnail links to the full image.
    - File List Section: Display non-image files as links.
    - Includes "Print Details" button.

- **File Handling Details:**
  - PHP upload handler: Use `wp_handle_upload()`, check for errors, move file to `uploads/PrTr_files/{project_id}/` (consider subfolders per project). Record path, original name, MIME type, uploader, timestamp in `PrTr_project_files`.
  - Deletion: Requires AJAX action. Handler verifies user permission, deletes file from server (`wp_delete_file()`), deletes record from `PrTr_project_files`. Update file list in Add/Edit modal via JS.
- **Image Display:**
  - Details Modal (PHP/JS): Query `PrTr_project_files` for associated files. If image MIME type, generate `<img>` tag with `src` pointing to file path, scaled via CSS. Wrap `<img>` in `<a>` tag linking to the full image path.
  - Lightbox (JS): Integrate a simple JS lightbox library. Attach click listener to the image links (`<a>` tags) to open the full image in the lightbox.
- **Printing:**
  - Details Modal: "Print Details" button triggers `window.print()` via JS.
  - CSS (`PrTr_style.css`): Use `@media print` rules to:
    - Hide non-content elements (modal backdrop/chrome, close button, print button itself).
    - Ensure modal content (`#pt-details-modal .pb-modal-content` or inner container) expands to page width.
    - Optimize layout for 8.5x11 paper (adjust font sizes, margins, remove unnecessary backgrounds/shadows).
    - Ensure all relevant data is visible and not cut off.

6. User Access Control (Role-Based)

- **Required Roles:** These roles must be created in WordPress.
  - `view_projects` (Display Name: "View Projects")
  - `edit_projects` (Display Name: "Edit Projects")
  - `full_project_access` (Display Name: "Full Project Access")
- **Implementation Logic:**
  - **Data Query (`PrTr_get_projects_ajax_data`):**
    - Get current user's roles.
    - If user has `full_project_access`, show all projects.
    - Otherwise, get departments associated with user's roles (similar to Property Book logic). Add `WHERE p.department_id IN (...) OR p.client_department IN (...)` to the SQL query.
  - **Add Button Visibility (PHP Shortcode):** Only render if `current_user_can('edit_projects')`. *(Note: `current_user_can` checks for role existence)*.

    - **Edit Button Visibility (JS - DataTables Render):** Check localized `can_edit` flag (set via PHP `wp_localize_script`). Flag is true if user has `edit_projects`. Additionally, the button's click handler (or the POST handler) must verify if the specific project's `department_id` allows editing by this user OR if user has `full_project_access`.
    - **POST Handler (PHP - Add/Edit Save):**
        - Check nonce.
        - Check if user has `edit_projects` role using `current_user_can('edit_projects')`. Abort if not.
        - For **updates**, also check if user has `full_project_access` OR if the `department_id` of the project being updated matches one of the user's associated departments. Abort if not permitted.
    - **Filter Dropdowns (PHP):** Filter 'Department' dropdown options based on user's associated departments unless they have `full_project_access`.

7. Security Considerations

- Strict adherence to Code Ruleset 1 security section [cite: 7-17, 836-845] (Nonces, Sanitization, Escaping, Permissions/Roles, `$wpdb->prepare`, Direct Access Prevention).
- Secure file upload handling (use WordPress functions, validate).
- Ensure file deletion logic correctly verifies user permissions for the specific project.

8. Known Issues / Future Work

- *(To be populated during development)*
- AJAX form submission is out of scope due to previously identified server-level issues. Standard POST will be used.

9. Dev Log

- **v0.1 (2025-04-15):** Initial Development Document creation.

Overall Status:

- The initial planning and setup phase for the Project Tracker WordPress plugin is complete.
- Development work on the core functionality is ready to begin.

Key Accomplishments / Current State:

1. **Development Document Finalized:** The "Project Tracker - Development Document v0.1" has been created, reviewed, and finalized, outlining the project scope, requirements, database structure, functionality, and user access control.
2. **Companion Document Established:** The document clearly notes its dependency on the "Property Book Development Document" (from previous context) for Code Ruleset 1 and established development patterns.
3. **Database Setup:**
    - The SQL `CREATE TABLE` statements for all necessary custom tables (`PrTr_projects`, `PrTr_project_categories`, `PrTr_project_statuses`, `PrTr_project_files`) have been defined, reviewed, and executed.
    - The `PrTr_project_statuses` table has been populated with the defined statuses and their sort order.
    - The `PrTr_project_categories` table has been populated with the initial brainstormed list of categories.
4. **Roles Defined:** The required WordPress user roles (`view_projects`, `edit_projects`, `full_project_access`) and their associated permissions within the plugin have been defined. (These roles need to be created in WordPress if not already done).
5. **File Structure Defined:** The core plugin file structure (`PrTr_init.php`, `PrTr_project.js`, `PrTr_style.css`) and the upload directory (`wp-content/uploads/PrTr_files/`) have been specified [cite: 41-45, 66, 67].
6. **Core Decisions Made:** Key decisions like using standard POST for data submission, the approach for image thumbnails (CSS scaling + lightbox), and filtering requirements are documented.

Next Steps:

- According to the development document, the next logical step is to begin the initial plugin setup within `PrTr_init.php` (e.g., basic plugin header, including files, defining constants, enqueuing scripts/styles, registering the shortcode).

---

v0.1.5 (2025-04-15)

Project Tracker - Development Log

Date: April 15, 2025

Version Status at End of Day: Approximately v0.1.5 (PrTr_init.php logic complete)

Summary of Progress:

The focus of today's session was the initial creation and implementation of the core PHP logic within the `PrTr_init.php` file for the Project Tracker plugin. Development adhered strictly to Code Ruleset 1 and utilized established patterns from the Property Book plugin where applicable.

Key Accomplishments:

1. **`PrTr_init.php` Structure:** Established the complete file structure, including plugin header, constants definition (`PRTR_PLUGIN_PATH`, `PRTR_PLUGIN_URL`, `PRTR_UPLOAD_DIR_NAME`), and standard WordPress security checks (`defined('ABSPATH')`) [sources: 1-7].
2. **Script & Style Enqueueing (`prtr_enqueue_scripts`):** Implemented the function to enqueue `PrTr_style.css` and `PrTr_project.js`, along with jQuery UI Datepicker CSS/JS. Correctly localized essential data for JavaScript (`ajax_url`, various nonces, user permission flags) using `wp_localize_script` [sources: 8-19].
3. **Shortcode (`prtr_project_tracker_shortcode`):** Implemented the main `[project_tracker_list]` shortcode handler. This function checks user login status and permissions, displays feedback messages based on URL parameters, and renders the primary HTML structure (using `?>...<?php` syntax per ruleset) including containers for filters, action buttons, the main DataTable (`#pt-projects-main-table` with headers), and modal placeholders (`#pt-project-modal`, `#pt-details-modal`) [sources: 20-49]. Calls were added to placeholder functions to render modal content.
4. **AJAX Handlers:**
    - `prtr_get_projects_ajax_data`: Fully implemented to handle AJAX requests for the main table data. Includes nonce/permission checks, role-based data filtering (using `prtr_get_allowed_department_ids_for_user`), database queries with necessary JOINs, data formatting (dates, currency, boolean), default sorting, and generation of action buttons [sources: 51-85].
    - `prtr_get_project_details`: Fully implemented to handle AJAX requests for the details modal. Includes nonce/permission checks (including specific project view check based on department), database queries for project details and associated files (`PrTr_project_files`), data formatting, and construction of file download URLs [sources: 85-118].
5. **POST Handler (`prtr_handle_project_save`):** Fully implemented the function (hooked to `init`) to process Add/Edit form submissions. Includes checks for request method/action, nonce verification, user permission checks (general edit role and specific project edit permission), sanitization of all relevant `$_POST` fields using WordPress functions, basic validation logic, file deletion handling (checking ownership, deleting server file via `wp_delete_file`, deleting DB record), file upload handling (using `wp_handle_upload` with a custom directory filter, storing results), database

INSERT/UPDATE logic using `$wpdb`, and redirection back to the project page with status messages [sources: 148-224].

6. **Modal Rendering Functions:**
    - `prtr_render_project_form`: Implemented to generate the complete HTML form structure for the Add/Edit modal, including all fields, fieldsets, dynamically populated & filtered dropdowns, file upload input, existing file list with delete checkboxes, and handling for pre-filling data in edit mode [sources: 318-378].
    - `prtr_render_project_details`: Implemented to generate the static HTML structure for the Details modal, mirroring the form layout and providing `<span>` placeholders for data population via JavaScript [sources: 380-401].
7. **Helper Functions:** Implemented all necessary helper functions: `prtr_get_allowed_department_ids_for_user` (role-to-department mapping), `prtr_generate_action_buttons` (dynamic button generation based on permissions), `prtr_get_project_column_formats` (for DB operations), `prtr_custom_upload_directory` (for file uploads), and `prtr_redirect_with_status` (using the hardcoded redirect URL) [sources: 119-147, 224-230, 232-244].
8. **Clarifications:** Confirmed that DataTables requires custom JavaScript initialization (disabling automatic behavior) and addressed all identified TODO comments related to core logic implementation. The only remaining TODO is an advisory note regarding optional file download security hardening [source: 116].

Current State:

- The `PrTr_init.php` file [source: 1-401] is functionally complete based on the documented requirements. All PHP logic for data handling, display structure, AJAX requests, and form processing is implemented.

Next Steps:

1. Create the `PrTr_project.js` file.
2. Implement the JavaScript functionality within `PrTr_project.js`, starting with:
    - Basic file structure (`jQuery(document).ready(...)`).
    - Custom DataTables initialization for `#pt-projects-main-table`, configuring columns to match PHP headers, setting the AJAX source to `prtr_get_projects_ajax_data`, and disabling default features where necessary.
    - Event handlers for buttons ("Add New", "Edit", "Details", "Print", "Clear Form").
    - AJAX call logic to fetch details and populate modals.
    - Modal show/hide logic.
    - Lightbox integration for images.
3. Create the `PrTr_style.css` file and add necessary CSS rules for layout, styling consistency with Property Book, and print view optimization.

PHP (`PrTr_init.php`)

- **Plugin Activation:** The plugin activates without PHP fatal errors.
- **Core Setup:** Constants (`PRTR_PLUGIN_PATH`, etc.) are defined. File structure placeholders are noted. Direct access is prevented.
- **Script/Style Enqueueing (`prtr_enqueue_scripts`):**
    - Function exists and is hooked (currently with the `has_shortcode` check bypassed for testing).
    - Enqueues DataTables Core, Buttons, HTML5 Export, pdfmake, vfs_fonts CSS/JS from CDN (mirroring Property Book).
    - Enqueues jQuery UI Datepicker CSS/JS (Project Tracker specific).
    - Enqueues the plugin's custom `PrTr_style.css` and `PrTr_project.js`.
    - Correctly localizes data (`prtr_plugin_data`) including AJAX URL, nonces, and permission flags.
- **Shortcode (`prtr_project_tracker_shortcode`):**
    - Registered for `[project_tracker_list]`.
    - Checks for user login and base view/edit/full permissions.
    - Displays feedback messages after POST submissions.
    - Renders the main HTML structure: filter placeholder, action buttons container (with conditional "Add New Project" button), table container with `<table>` shell and correct `<thead>`, and modal shells (`#pt-project-modal`, `#pt-details-modal`).
- **Modal Content Rendering:**
    - `prtr_render_project_form`: Renders the Add/Edit modal's form structure, including fieldsets, labels, inputs, dropdowns (dynamically populated & filtered by role), file input, and existing file list display (with delete checkboxes).
    - `prtr_render_project_details`: Renders the Details modal's static HTML structure with spans for data population.
- **AJAX Handlers:**
    - `PrTr_get_projects_ajax_data`: Fetches raw project data (including `project_id`, `category_id`, `status_id`, `department_id`, `updated_by` username, joined names for department/category/status), applies role-based filtering, checks for DB errors, and returns data correctly formatted as `{"data": [...]}` using `wp_send_json`. (Successfully populates the table via JS).
    - `PrTr_get_project_files_ajax`: Fetches the file list for a specific project ID, joins for uploader display name, formats the upload date (date only), correctly generates download URLs, checks for errors, and returns data via `wp_send_json_success(['files' => ...])`. (Successfully populates file list in Details modal).
- **POST Handler (`prtr_handle_project_save`):**

    - Handles form POST submissions.
    - Verifies nonce and permissions (including edit permission for specific project on updates).
    - Sanitizes and validates input data.
    - Correctly handles saving a blank "Location" field (results in `NULL` or empty string, not "0").
    - Handles file uploads using `wp_handle_upload` and the custom directory filter.
    - Handles file deletions (server file and DB record).
    - Performs `$wpdb->insert` or `$wpdb->update` correctly.
    - Redirects back to the page with appropriate status messages.
- **Helper Functions:** `prtr_get_allowed_department_ids_for_user`, `prtr_get_project_column_formats`, `prtr_custom_upload_directory`, `prtr_redirect_with_status` are implemented.

JavaScript (`PrTr_project.js`)

- **Loading:** Loads correctly (when `has_shortcode` check is bypassed).
- **DataTables Initialization:**
    - Initializes DataTables on `#pt-projects-main-table`.
    - Successfully fetches data via the `PrTr_get_projects_ajax_data` action.
    - Populates table rows.
    - Uses JavaScript `render` functions to correctly format display values for dates, currency, and booleans ('Yes'/'No').
    - Uses a JavaScript `render` function to display "Details" and conditional "Edit" buttons in the Actions column.
    - Basic DataTables features (search, pagination, info, length change) are active.
- **Add New Modal:**
    - "Add New Project" button click handler works.
    - Opens the `#pt-project-modal`.
    - Clears the form correctly using `reset()` and clears hidden ID.
    - Sets the modal title to "Add New Project".
    - "Clear Form" button inside the modal works.
    - Modal closes correctly via 'X' button and backdrop click.
- **Edit Modal:**
    - "Edit" button click handler works.
    - Retrieves row data from DataTables instance (`rowData`).
    - Populates the modal form fields correctly, **including all dropdowns**.
    - Sets the modal title to "Edit Project (...)".
    - Populates the "Last updated by..." footer text (currently showing **username** and date correctly).
    - Shows the modal.
- **Details Modal:**

    - "Details" button click handler works.
    - Populates main project fields from `rowData` correctly (Location field shows blank when appropriate).
    - Successfully fetches the associated file list via the separate `PrTr_get_project_files_ajax` call.
    - Displays the file list correctly: renders image thumbnails, links for other files, and the associated meta text (Uploader **Display Name** and **Date Only**, with no comment artifacts).
    - Sets the modal title correctly.
    - Populates the "Last updated by..." footer text correctly (showing **username** and date only).
    - Shows the modal.
    - Modal closes correctly via 'X' button and backdrop click.
- **Print Button:** Exists in the Details modal; `window.print()` is triggered on click.

CSS (`PrTr_style.css`)

- File created.
- Contains base modal styles copied from Property Book.
- Contains placeholder rules and basic print styles.

NOTE: Any txt files provided are only listed as txt for uploading purposes. The real files in the projects have the exact same content and are the appropriate file types.

Example: “PrTr_init.php.txt” is “PrTr_init.php” in the real project directory.

---

v0.2.0 (2025-04-17)

Status & Accomplishments:

We reached a stable point after implementing and debugging several key features:

1. **File Handling:**
    - Successfully implemented file uploads into project-specific, timestamped sub-directories (`PrTr_files/{project_id}_{Ymd}/`).
    - Added a dynamic list of existing files (with image thumbnails) to the Edit modal.
    - Implemented file deletion via checkboxes in the Edit modal, including a JavaScript confirmation alert.

    - Incorporated database transactions into the project save/file deletion process for data consistency.
2. **Data Display & Consistency:**
    - Fixed the display of "Client Department" name in both the Details modal and the main table view.
    - Corrected the default sort order for the main table to use Status Order (asc) then Last Updated (desc).
    - Resolved various DataTables warnings related to data parameters (`value`, `expenses`, `null` for actions).
    - Ensured the main AJAX query (`PrTr_get_projects_ajax_data`) fetches all necessary project data (`p.*`) plus joined fields, aligning with the Property Book strategy for future flexibility.
3. **Permissions:**
    - Refined the logic for the "Edit" button in the Actions column, ensuring it only displays when the logged-in user has the specific rights to edit *that particular project* based on roles and ownership (not just client relationship). Confirmed this is working as expected.
4. **Styling & Layout:**
    - Synchronized the CSS for the table container, table cells, action buttons bar, and DataTables controls (length menu, info, pagination) with the Property Book plugin for visual consistency.
    - Adjusted table header CSS to allow wrapping, improving layout manageability.
    - Reverted modal layouts to a single column based on user preference.
    - Removed potential text-shadows for better text clarity.

New Version Number:

Let's assign **v0.2.0** to this stable state. This reflects the completion of the initial setup, core data handling, modals, file management basics, permissions, and initial styling synchronization.

Development Document Updates:

Here are the suggested updates for your `Project Tracker - Development Document (1).docx` file. You can incorporate these before our next session.

1. **Add a new section (e.g., Section 10):**

```
Unset

10. Development Log
```

v0.1 (2025-04-15): Initial Development Document creation. Database schema defined and implemented. Roles defined. Core file structure specified.

v0.2.0 (2025-04-17): Completed initial PHP logic (PrTr_init.php v0.1.5), JS logic (PrTr_project.js v0.1.5), and CSS setup (PrTr_style.css v0.1.1).

- Implemented core shortcode display, AJAX data fetching for main table.

- Implemented Add/Edit modal (POST submission) and Details modal (AJAX population).

- Implemented file uploads to project-specific, timestamped sub-directories (wp-content/uploads/PrTr_files/{ID}_{Ymd}/).

- Added dynamic existing file list (with thumbnails) and delete checkbox functionality (with confirmation) to Edit modal.

- Implemented database transactions for project save/file deletions.

- Fixed Client Department display in modals and main table.

- Corrected default table sorting (Status Order asc, Updated Date desc).

- Refined Edit button visibility based on specific user permissions per project row.

- Synchronized basic CSS with Property Book styles; resolved DataTables warnings.

- Next Steps: Implement main table filtering.

2. **Update Section 5 - File Handling Details:**
    - Modify the description of the upload path [source: 1509] to match the implementation:

      PHP upload handler: Use `wp_handle_upload()`, check for errors, move file to `uploads/PrTr_files/{project_id}_{Ymd}/` (where Ymd is the upload date).

3. **Update Section 6 - User Access Control:**
    - Replace the text describing Edit button visibility / POST handler checks [source: 1527, 1529] with the refined wording:

      Edit Button Visibility (JS - DataTables Render) / POST Handler (PHP - Add/Edit Save): An 'Edit' button is visible if the user has the `edit_projects` role. However, clicking 'Edit' or submitting an edit via the form requires further validation:

        - *A user with `full_project_access` (and `edit_projects`) can edit any project.*
        - *A user with `edit_projects` but without `full_project_access` can only successfully edit projects where the project's **Owning Department** (`department_id`) matches one of the departments associated with the user's roles. They cannot edit projects solely based on being in the **Client Department**.*
        - *(POST Handler Specific): Check nonce. Check if user has `edit_projects` role. Abort if not. For updates, perform the specific permission check described above. Abort if not permitted.*
4. **Review Section 8 - Known Issues / Future Work:**
    - Ensure this section is up-to-date. AJAX form submission is still out of scope [source: 1533]. The main next item is implementing filtering [source: 1484-1489]. Lightbox for images is also pending [source: 1515].

---

v0.2.1 (2025-04-17) - Stable Baseline

This version marks the completion of the initial core functionality, data handling, modals, file management basics, permissions, sorting, and styling synchronization. The plugin is considered stable at this point.

Key Accomplishments / Features:

- **Core Structure:** Implemented main shortcode `[project_tracker_list]` display, AJAX data fetching (`p.*` strategy adopted), Add/Edit modal (standard POST submission), Details modal (AJAX population), file upload handling, and basic print view structure.
- **Data Display:** Main table loads and displays project data including joined names (Owner Dept, Client Dept, Category, Status). Default sorting correctly implemented (Status Order asc, Last Updated desc). Configurable column visibility via JS `columnDefs` and CSS classes is set up (Description column added, visibility rules defined).
- **File Handling:** File uploads save correctly to project-specific, timestamped sub-directories (`wp-content/uploads/PrTr_files/{ID}_{Ymd}/`). Edit modal dynamically loads the existing file list via AJAX, displaying image thumbnails. File deletion works via checkboxes in the Edit modal with a JS confirmation prompt.
- **Data Integrity:** Database transactions implemented for project record saving and file record deletions. Correct handling of `NULL` values implemented for 'Value' and 'Expenses' fields (saved as `NULL`, displayed as blank in table and modals).
- **Permissions:** Edit button visibility correctly controlled based on user role (`edit_projects`) and specific project edit permissions (`can_edit_this` flag passed via AJAX). Role-based filtering applied to Department dropdown population in filter UI stub.
- **Styling:** CSS synchronized with Property Book where applicable (modals, table styles, action buttons, DataTables control wrappers). Modals reverted to single-column layout. Text clarity improved (text-shadow removed).

Discovered Issues & Fixes During Development:

- **File Upload (New Projects):** Initial implementation skipped uploads for new projects due to `$project_id` not being available before the upload logic ran. Fixed by ensuring `$project_id` (from `$wpdb->insert_id`) is set before attempting uploads post-transaction.
- **Edit Modal File List:** Existing files list wasn't rendering because the PHP condition (`if ($is_edit_mode)`) only ran on initial page load. Fixed by removing PHP rendering and implementing dynamic loading via a separate AJAX call within the Edit button's JS handler.
- **DataTables Warnings:** Encountered "unknown parameter" warnings for `value`, `expenses`, and `null` (actions column). Resolved by adding `defaultContent` and

making `render` functions more robust for null/undefined data in JS `columns` definitions, and changing `data: null` to `data: 'project_id'` for the Actions column.
- **Incorrect Default Sorting:** Table was sorting incorrectly (by Public Release/Expenses). Fixed by adding a hidden `status_order` column definition to JS and correcting the `order` option in DataTables initialization, removing a duplicate/incorrect `order` definition.
- **Value/Expenses `NULL` Display:** Modals were displaying `$0.00` for `NULL` values. Fixed by updating the specific JS functions that populate the Edit and Details modals to check for `null` and output empty strings (`''`).
- **Filter Visibility:** Filter container remained hidden due to a persistent inline `style="display: none;"` in PHP overriding the CSS `display: flex;`. Fixed by removing the inline style from the `prtr_render_filter_controls` function.
- **CSS Conflicts:** Addressed minor CSS issues like modal layout responsiveness (reverted to single column) and text clarity (removed text-shadows).

Updated Workflow Practices / Do's & Don'ts:

- **Data Fetching:** Main table AJAX (`PrTr_get_projects_ajax_data`) **will** fetch all project table columns (`p.*`) plus necessary joined names/sorting fields, consistent with Property Book, allowing display flexibility via JS/CSS. Related data (e.g., files) will be fetched on demand via separate AJAX calls for modals.
- **Column Visibility:** Use the established 3-part method (JS `columns` definition, JS `columnDefs` setting `visible` or `className`, CSS hiding rules for `.pt-col-always-hidden` / `.pt-col-desktop-only`) for managing table column display.
- **`NULL` Handling:** Explicitly handle potential `NULL` values from the database in both PHP saving logic (saving `NULL` when input is blank) and JavaScript display logic (checking for `null` before formatting or displaying).
- **Transactions:** Use database transactions (`START TRANSACTION`, `COMMIT`, `ROLLBACK`) for operations involving multiple related database writes (e.g., project save + file deletions) to ensure data consistency.
- **Code Updates:** Continue providing code updates one logical step at a time, ensuring clear context or providing full function/file replacements when substantial changes occur. Prioritize code readability and maintain indentation.
- **Debugging:** Use browser console (`console.log`) for client-side JS debugging (tagging messages with `//--console debug line` for easier cleanup). Use server-side `error_log` for PHP debugging when needed.

Scope Changes:

- **File Upload Path:** Confirmed and implemented upload path as `wp-content/uploads/PrTr_files/{project_id}_{Ymd}/` (differs slightly from initial dev doc reference to only `{project_id}`).
- **Filters:** Confirmed the specific list of filters required for the next phase (Owner Dept, Client Dept, Category, Status, Public Release Checkbox, Date Range). Clarified date range logic.

Next Steps:

1. **Lightbox Implementation:** Add lightbox functionality for images shown in the Details modal.
2. **Creat download buttons and Logic:** As with the Property Book Plugin, this project still needs the buttons added to download the filtered table. This should mirror the “Download Current View” button from the Property Book. This project does not need a hand receipt button, but will need individual projects to be printable. That is handled on the details modal.
3. **Print View Optimization:** Further refine print styles if necessary after core functionality is complete.

v1.0 (2025-04-21): Stable Release Candidate Finalization.

- Completed feature implementation based on v0.2.1 next steps.
- Fixed "Last updated by" display name in Details modal footer to show user display name instead of username.
- Implemented image lightbox functionality (using basicLightbox library) for image thumbnails in the Details modal.
- Added "Download Current View" button using DataTables Buttons extension for PDF export of the current table view, mirroring Property Book functionality.
- Reworked Details modal "Print Details" function: replaced CSS print styles with a JavaScript iframe method to generate a clean, dedicated print view of modal content, including thumbnails.
- Resolved outstanding questions regarding ruleset interpretation, database conventions, file paths, and feature implementation details; added Section 10 (FAQ & Clarifications) to document these.
- Plugin considered feature-complete based on initial requirements for v1.0.

10: FAQ & Clarifications

This section addresses questions and confirms technical decisions made during development up to version 0.2.1.

Q: What is the correct table naming convention?
A: Table names prefixed with `PrTr_` (e.g., `PrTr_projects`) are considered the full table names as provided by the user and are used directly without adding a WordPress database prefix.

Q: Is DataTables availability a prerequisite?
A: Yes, the DataTables JS library is a confirmed prerequisite for this plugin's functionality. It is explicitly enqueued by the plugin via `PrTr_init.php` and is not automatically applied by a separate generic DataTables plugin.

Q: How is user role mapped to department access?
A: Authorization for viewing or editing specific projects is determined by matching a user's WordPress role *display name* to a `department_name` in the `inv_departments` table. This is implemented in the `prtr_get_allowed_department_ids_for_user` helper function.

Q: How is `current_user_can()` used for permissions?
A: In this plugin, checks like `current_user_can('edit_projects')` are used solely to verify the *existence* of the specified role for the current user. They do *not* interact with the standard WordPress capability system, as project-specific authorization is handled via the role-to-department mapping logic.

Q: What is the final file upload path structure?
A: The confirmed and implemented upload path structure places files into project-specific, date-stamped subdirectories: `wp-content/uploads/PrTr_files/{project_id}_{Ymd}/`, where `{Ymd}` represents the date of the upload.

Q: How is file deletion handled?
A: Deletion of existing files is handled via checkboxes within the Add/Edit modal form. These selections are processed during the standard POST submission by the `prtr_handle_project_save` function. The previous mention of a separate AJAX deletion mechanism [cite: 97, 111] is superseded by this implemented POST-based approach.

Q: Is the Date Range filter logic correct?
A: Yes, the date range filtering logic currently implemented in `PrTr_project.js` is confirmed as meeting the project requirements.

Q: What is the plan for the Lightbox feature?
A: Before implementing the lightbox feature for image previews[cite: 115], investigation is needed to determine if the active WordPress theme (Aardvark) provides a suitable component. If not, a lightweight JavaScript library will be selected and integrated.

Q: Have the required WordPress roles been created?
A: Yes, the required WordPress roles (`view_projects`, `edit_projects`, `full_project_access`) have been confirmed as created and are available in the target WordPress environment.

Q: Which Code Ruleset should be followed?
A: Code Ruleset 1, as defined within this document (Project Tracker - Development Document), serves as the primary guideline. The current codebase (v0.2.1) reflects the most up-to-date workflow practices evolved during development.

Q: How should the Property Book code be referenced?
A: The complete codebase for the foundational Property Book plugin (v1.0 Stable) is included within the `Property Book Development.docx` document. This serves as a reference for established patterns and logic where applicable.

Q: Why are standard POST submissions used instead of AJAX for saving data?
A: Due to external server-level configurations or limitations that caused issues with AJAX POST requests during previous development phases (on the Property Book project), all form submissions involving data saving (Add/Edit Project) utilize standard HTML POST requests with page redirects. AJAX is used successfully for data retrieval (populating tables/modals).

Public View Addition

11. Public View Addition Plan (v1.1 Target)

11.1. Overview & Goal

This section outlines the plan for implementing a new public-facing shortcode, tentatively `[pt_public_projects]`, for the Project Tracker plugin. The goal is to display projects marked for public release in a visually engaging format (cards) suitable for website visitors, showcasing the City of London's work effectively. The implementation will prioritize efficient loading and responsiveness, especially considering potentially hundreds of projects. It will utilize a WordPress REST API endpoint for data fetching and JavaScript for dynamic rendering.

11.2. Core Technology & Approach

- **Data Source:** WordPress REST API endpoint (publicly accessible).
- **Data Loading:** Asynchronous JavaScript and XML (AJAX) requests initiated by client-side JavaScript.
- **Rendering:** Client-side JavaScript dynamically generating HTML for project "cards".
- **Layout:** Responsive card-based layout (CSS Grid or Flexbox).
- **Pagination:** "Load More" button functionality to load projects in batches.
- **Details Display:** On-demand modal window (adapted from the existing Details modal) showing project details and images upon card interaction.
- **Image Handling:** Cards will display a primary project thumbnail; the modal will display all associated images using the existing lightbox.

11.3. Backend Implementation (PHP)

1. **File Placement:** Code for the REST API endpoint registration and its handler function should be placed in `PrTr_functions.php` (ensure this file is included by `PrTr_init.php`) or a new dedicated PHP file included by `PrTr_init.php`.
2. **REST API Endpoint Registration:**
    - Register a new REST API route using `register_rest_route`.
    - **Route:** `/project-tracker/v1/public-projects` (Example)
    - **Methods:** `GET`
    - **Callback:** Point to the PHP handler function (e.g., `prtr_get_public_projects_rest_handler`).
    - **Permission Callback:** Use `__return_true` to make the endpoint publicly accessible (no authentication required).
    - **Arguments:** Define accepted query parameters (`args`) for pagination (`page`), filtering (`filter_dept`, `filter_cat`, `filter_status`), and sorting (`sort_by`, `sort_order`). Include validation and sanitization callbacks for these arguments.
3. **REST API Handler Function (`prtr_get_public_projects_rest_handler`):**
    - **Input:** Receives validated/sanitized parameters (`page`, filters, sort order) from the REST request object.
    - **Database Query (Projects):**
        - Construct a `WP_Query` or direct `$wpdb` query on the `PrTr_projects` table.
        - **WHERE Clause:** Filter strictly by `public_release = 1` AND apply any additional filters passed as parameters (Department, Category, Status).
        - **JOINs:** Join `inv_departments` (aliased for Owner and Client), `PrTr_project_categories`, and `PrTr_project_statuses` to retrieve necessary names and colors.
        - **SELECT Clause:** Select only necessary *public* fields: `project_id`, `display_id`, `owner_department_name`, `owner_department_color`, `client_department_name`, `client_department_color`, `category_name`, `status_name`, `scheduled_start_date`, `expected_completion_date`, `project_completed_date`, `location`, `description`. *Exclude:* `value`, `expenses`, `comments`, `associated_purchase_orders`, `updated_by`, internal IDs (`department_id`, `category_id`, etc.) unless needed for linking.
        - **ORDER BY:** Apply sorting based on the `sort_by`/`sort_order` parameters (e.g., `updated_at DESC`, `status_order ASC`, `scheduled_start_date ASC`). Default sort if none provided.

        - **Pagination:** Implement pagination using `LIMIT` and `OFFSET` logic based on the requested `page` number and a defined `posts_per_page` value (e.g., 12). Calculate the `OFFSET`.
    - **Database Query (Primary Images):**
        - After getting the batch of project IDs from the first query, perform a *second* efficient query on `PrTr_project_files`.
        - For each `project_id` in the current batch, retrieve the `file_path` (or necessary info to construct the URL) of the *first* associated image file (`WHERE file_type LIKE 'image/%'`). Group by `project_id` and use `MIN(file_id)` or similar logic if creation order isn't reliable.
    - **Data Structuring:** Combine the project data and the primary image URL data into a structured array. Calculate total number of public projects (for pagination total) via a separate count query if needed.
    - **JSON Response:** Return the data using `wp_send_json_success` containing the array of project objects for the current page and pagination info (e.g., `currentPage`, `totalPages`). Handle potential errors with `wp_send_json_error`.

11.4. Frontend Implementation (JavaScript & Shortcode)

1. **Shortcode Output (`[pt_public_projects]` Handler):**
    - The PHP function handling this shortcode (e.g., `prtr_public_projects_shortcode` defined in `PrTr_functions.php` and registered) should output:
        - Optional filter controls (dropdowns for Department, Category, Status; sort dropdown).
        - A main container `div` (e.g., `<div id="pt-public-project-cards"></div>`) where cards will be rendered.
        - A "Load More" button (`<button id="pt-load-more-projects">Load More</button>`), initially hidden or disabled.
        - A loading indicator element.
2. **JavaScript Logic Placement:** Add necessary JS code to `PrTr_project.js` or create a new, conditionally enqueued JS file (e.g., `PrTr_public.js`) specifically for this shortcode. Ensure dependencies (like jQuery) are met.
3. **Initial Data Load:**
    - On document ready (or equivalent), if the `#pt-public-project-cards` container exists:
        - Initialize variables for the current page (starts at 1), filters (empty), sort order (default).

        - Make an initial AJAX `GET` request to the REST API endpoint (`/project-tracker/v1/public-projects`) with `page=1` and default sort/filter parameters.
        - Show loading indicator.
4. **Card Rendering (`renderCards` function):**
    - Create a reusable JS function that takes an array of project objects.
    - This function loops through the projects and generates the HTML string for each card (`<div class="project-card">...</div>`).
    - Card HTML should include: Primary image (`<img>` tag using the fetched URL), Project Title/ID, Department (styled with color), Status, Category, possibly a short description snippet, and store the `project_id` in a data attribute (e.g., `data-project-id="..."`).
    - Append the generated HTML cards to the `#pt-public-project-cards` container.
    - Hide loading indicator.
5. **"Load More" Button Logic:**
    - Attach a click event listener to `#pt-load-more-projects`.
    - On click:
        - Show loading indicator near the button.
        - Increment the current page variable.
        - Make an AJAX `GET` request to the REST endpoint, passing the *new* page number and any active filters/sort order.
        - On success: Append the newly rendered cards (using the `renderCards` function) to the container. Update the button's state (disable/hide if it was the last page). Hide loading indicator.
6. **Filtering/Sorting Logic (Optional - Phase 2?):**
    - Attach change event listeners to filter/sort dropdowns.
    - On change:
        - Update the filter/sort variables.
        - Reset the current page variable to 1.
        - Clear the `#pt-public-project-cards` container.
        - Show loading indicator.
        - Make a new AJAX `GET` request to the REST endpoint with the *new* filter/sort parameters and `page=1`.
        - Render the results using `renderCards`.
7. **Card Click / Modal Logic:**
    - Attach a click event listener (using delegation on the container) to `.project-card`.
    - On click:
        - Retrieve the `project_id` from the card's data attribute.

    - Make an AJAX `POST` request to the *existing* `PrTr_get_project_files` action handler (ensure nonce isn't required or handle appropriately if adapting for public use - ideally, the REST endpoint could optionally return all image URLs).
    - Prepare the necessary project detail data (likely already available from the data used to render the card).
    - Create/Select the public details modal element (`#pt-public-details-modal`).
    - Populate the modal with project details (excluding sensitive info) and the fetched file list (displaying images, potentially triggering the lightbox).
    - Show the modal.

11.5. Styling (CSS)

1. **File:** Add new rules to `PrTr_style.css`.
2. **Card Layout:** Define styles for `.project-card` using CSS Grid or Flexbox to create a responsive multi-column layout. Add padding, margins, borders, box-shadows for visual appeal.
3. **Card Content:** Style the primary thumbnail (`img`), project title (`h3`/`h4`), details text (`p`, `span`), and department color accents. Ensure consistent typography and spacing.
4. **Interactions:** Add `:hover` styles for cards.
5. **Filters/Sort:** Style dropdowns if implemented.
6. **Load More:** Style the button.
7. **Modal:** Define styles for `#pt-public-details-modal` (can likely reuse/adapt `.pb-modal` and related styles, but ensure no admin-specific styles are included). Style the content display within the modal.
8. **Responsiveness:** Ensure cards, filters, and modals adapt well to different screen sizes using `@media` queries.

11.6. Data Considerations

- **Excluded Fields:** Explicitly exclude `value`, `expenses`, `comments`, `associated_purchase_orders`, `updated_by`, internal IDs (like `department_id`, `category_id`, etc., unless essential for linking logic not exposed to the user) from the data returned by the REST API and displayed on the frontend.
- **JSON Structure:** Define the expected JSON format from the REST endpoint (e.g., `{ success: true, data: { projects: [ { project_id: ..., display_id: ..., owner_department_name: ..., owner_department_color: ..., primary_image_url: ..., ... } ], pagination: { currentPage: ..., totalPages: ... } } }`).

11.7. Development Considerations

- **"Wow" Factor:** Implement smooth loading transitions (e.g., fade-in for cards), use high-quality thumbnails where possible, and focus on clean, modern CSS design.
- **Performance:** Optimize database queries in the REST handler. Ensure efficient image loading (only primary thumbnail initially). Test performance with a large number of simulated projects.
- **Code Re-use:** Leverage existing AJAX handlers (`PrTr_get_project_files_ajax`) and modal structures/styling where appropriate, adapting them for public use.
