ChatGPT

Debugging Apollo Events Portal: PHP Static Output & System Improvements

Provided Code Context

Below are two code blocks for analysis:

1. Printed HTML Output (from Localhost browser):

```
<!-- Paste the rendered HTML output of the portal page here -->
```

2. PHP Source Code (`portal-discover.php`):

```
<?php
// Paste the contents of portal-discover.php here
?>
```

(The first block is the actual HTML that was output in the browser, and the second is the PHP template source. The goal is to compare these to identify why the PHP dynamic parts aren't functioning as expected.)

Analysis: PHP Not Executing Dynamic Content

Carefully compare the **printed HTML output** with the **PHP source code** to determine why the PHP is not fetching or displaying dynamic data from the database. Key points to consider:

- **Missing Dynamic Data:** Identify sections in the PHP code that are supposed to display dynamic content (e.g. event listings, DJ names, locations) but appear **hard-coded or empty** in the HTML output. For example, check if event details like DJ list or location are present in the PHP code but not showing up in the rendered HTML.

- **PHP Execution vs Static HTML:** Determine if the PHP template is being executed at all or if it's outputting static HTML. Look for signs that PHP logic might have been bypassed. For instance, the code has `defined('ABSPATH') || exit;` at the top – ensure the template is actually running within WordPress (if `ABSPATH` wasn't defined, the PHP would exit and yield no dynamic content). Also confirm that the page using this template (likely the "Eventos" page) is correctly set up to use the plugin's template in a WordPress context.

- **Transient Caching Effects:** The PHP uses a transient (caching) for events (see any usage of `get_transient` or `set_transient`). If the transient is not updating or if no new query is run, the events list might not populate as expected. Check if the transient logic might cause **stale or empty data** to be used. For example, if the transient was set on a prior day or not cleared, the page might show an outdated or no events. Ensure that when no transient exists (or

after expiration), a fresh query runs. If the transient is not the culprit, consider other query issues.

- **WordPress Query Results:** Verify whether the events query in the PHP code is actually retrieving events. The HTML output shows some event cards (if any are listed). If **no events are listed** in the output (or it shows a "Nenhum evento encontrado" message), it means the query returned no posts. This could be due to query filters (date, post type, status) that don't match existing events. Compare the query parameters in PHP (post type, meta_query for dates, etc.) with the data in your database to ensure there should be matching events. If events exist in the DB but aren't appearing, the query logic may need adjustment.

- **Unexecuted PHP Code Blocks:** Look at any PHP code embedded in the HTML (e.g., `<? php ... ?>` blocks for dynamic values). In the HTML output, these should be **replaced by actual data**. If you see placeholders or empty spans where PHP output should be, that indicates those PHP blocks ran but perhaps returned empty strings (e.g., an empty `$dj_display` resulting in "Line-up em breve" text). However, if the PHP code itself (like raw `<?php ?>` tags or PHP logic) appears in the output (which would be unusual), it means PHP isn't being interpreted at all. Based on the provided output, it seems the template did run (since some events are listed), but certain dynamic parts are empty or defaulted. We need to pinpoint why those parts aren't populating.

By analyzing these differences and clues, we will identify the root cause of why the PHP template is not fully executing as expected or why it's not retrieving data properly.

Code Structure and Plugin Context

Examine the **structure of the PHP code** and any references to other components to understand how this event portal system is designed:

- **Template Role:** This `portal-discover.php` appears to be a template file for the Apollo Events Manager plugin, likely controlling the front-end display of an events "portal". It is meant to be used on the "/eventos/" page (as indicated by the comment about **STRICT MODE** usage for this path). Verify that WordPress is loading this template for the correct page. (Often, custom post types or plugins override the page template; ensure that's configured correctly so this PHP runs.)

- **Custom Post Type & Taxonomies:** The code references a post type `event_listing` and taxonomies like `event_listing_category` and `event_sounds`. This suggests the plugin defines a custom post type for events (possibly using or emulating a plugin like WP Job Manager or a custom events system). Understand that events are stored as posts of type `event_listing` with custom fields (meta) such as `_event_start_date`, `_event_location`, `_event_banner`, `_timetable`, etc. The presence of these meta keys and the logic to retrieve them means the plugin expects events to have those custom fields set. Similarly, DJs might be stored as separate posts (perhaps a "DJ" post type) referenced by IDs in the timetable, or just stored as text.

- **Data Flow & Dependencies:** Identify any external dependencies or functions:

- Functions like `get_header()` and `get_footer()` indicate this template integrates with the WordPress theme environment.

- The code heavily uses WordPress functions (`WP_Query`, `wp_get_post_terms`, `get_post_meta`, etc.), meaning it runs within WP. Ensure that the plugin or theme enqueues required scripts and styles (e.g., icon libraries like the `ri-...` classes, and possibly a custom JS for the modal).
- Check if there are references to any **JavaScript files or CSS** for the events portal. For example, there might be a `<script>` or `wp_enqueue_script` for a file named `apollo-events-portal.js` or similar (responsible for the lightbox and layout toggle). If such references are missing in the output, it implies they were not enqueued or printed.

- See if the template or plugin includes additional PHP files (like an AJAX handler or shortcode definitions). The mention of an **Apollo Event Modal** container (`<div id="apollo-event-modal">`) and an `apollo_modal_handler.php` (from provided files) hints that clicking an event is supposed to load details (maybe via AJAX). The system likely has an AJAX endpoint (`apollo_modal_handler.php`) that returns event details for the modal. Ensure we consider this in the fix for the lightbox functionality.

- **Plugin System:** This template is part of a larger plugin ("Apollo Events Manager"). It likely registers the custom post type and provides this template via a shortcode or direct template include. The class file `class-apollo-events-placeholders.php` (mentioned in repository) might manage placeholders or data injection. We should keep in mind the plugin's structure: templates, possibly a main plugin file (`apollo-events-manager.php`), and supporting scripts. Understanding this helps ensure our fixes integrate properly (e.g., if we need to enqueue scripts, it should be done via the plugin, not by hacking the template alone).

By reviewing these aspects, we gain insight into how data is supposed to flow from the database to the front-end, and what parts of the system need adjustments.

Debugging and Fixes

Based on the analysis, we will now **propose and apply corrections** to resolve the issues:

A. Fixing Database Queries & PHP Logic

- **Correct the Event Query:** Ensure the WP_Query in the PHP template retrieves upcoming events properly. In the provided code, the query filters by `_event_start_date >= now`. Potential issues:
- If events have start dates without times (e.g., stored as "YYYY-MM-DD" only), the comparison with a full datetime string could fail. Consider specifying the meta query `type` as DATE or using `date('Y-m-d')` for comparison to include events from today onward.
- If using current time with time portion, events that start later today might be included, but those with only date might be excluded. A fix is to use `'type' => 'DATE'` in the meta_query or adjust logic to be inclusive of the whole day.
- If the original code uses a transient (`apollo_upcoming_events_{date}`), make sure it refreshes when appropriate. For debugging, you might **disable the transient caching** to see fresh results (or reduce its expiration). Since we want to solve the "no data" issue, it might be simpler to remove the transient during development or ensure it's set to a short duration (like 5 minutes, which it is in code) and that `get_transient` is actually returning something.

- The **fixed approach** could be using a direct query without transient while debugging. For example, use `posts_per_page => -1` (or a reasonable limit) and confirm `post_status => 'publish'` is included so that only published events show. In the "fixed" code example, they

switched to `posts_per_page => -1` and explicitly set the meta query type to DATE – consider implementing that if appropriate.

- **Verify Query Results:** After adjusting the query parameters, test that events from the database are now returned. If `have_posts()` is still false (and you're certain events exist), try a simpler query (e.g., query just `post_type => event_listing` without date filter) to ensure the custom post type is recognized and populated. This helps isolate whether the issue is with the date filtering or something more fundamental (like the post type registration or the query not running at all).

- **Remove or Adjust Caching:** If caching is interfering, you can temporarily remove the transient usage:

- Bypass the `get_transient` check and always run a fresh `WP_Query` while debugging. Once it's working, you can reintroduce caching if needed, but ensure that it doesn't cache empty results unintentionally (for example, on a day with no events, the transient might store an empty query result; on the next day when an event is added, the transient key would change due to date, so that's probably okay).

- Alternatively, after fixes, consider caching by week or month if events list grows, and provide a way to clear cache when new events are published (maybe via WordPress hooks).

- **Error Handling:** The code has an error handling branch for `is_wp_error($events_query)`. If there was a DB error, it would log and show an error message. Ensure no errors are logged. If the query returns but has no posts, the code prints "Nenhum evento encontrado." If you saw that in output, it confirms the query ran but found nothing matching the criteria – reinforcing that we need to adjust the criteria.

By fixing the query logic as above, the PHP should correctly fetch and display event posts from the database.

B. Fixing Missing Variable Outputs (DJ List, Location, etc.)

- **DJ Names Not Showing:** In the current output, the DJ line shows **"Line-up em breve"** for events, meaning the `$dj_display` variable ended up empty (fallback text). We need to ensure actual DJ names appear when available:
- Check how event DJ information is stored. The code tries multiple meta fields (`_timetable`, `_dj_name`, `_event_djs`). Perhaps your events have DJs listed differently or not at all. For instance, maybe the events only had a text field for DJs that wasn't saved to `_dj_name` meta. Make sure to populate one of these fields for each event:
    - **_timetable**: If the event has a timetable (schedule) array with DJs, the code will use that. Ensure that meta exists and is an array of DJ entries. If using a plugin UI, maybe the timetable wasn't set up.
    - **_dj_name**: This might be a simple text meta for a DJ or headliner. If your events have a custom field for DJ names, save it as `_dj_name` so the code picks it up. The fixed code prioritizes `_timetable` then `_dj_name`. So adding a value to `_dj_name` for each event will make the DJ appear.
    - **_event_djs**: The original code considered an `_event_djs` meta possibly containing related DJ post IDs. If your system uses a relationship (each DJ is a post), you might need to ensure that meta is filled with the related DJ post IDs. If that meta wasn't being set, it

would also result in no DJs found. Since the fixed code you have did not include this third step (perhaps it was not needed or simplified), you can align with that simpler approach.

- **Code Corrections:** If you prefer, simplify the DJ retrieval logic:
  - If there's a **single DJ name field**, you can just use that. For example, if each event has a custom field "DJ(s)", use that directly instead of the complex loop.
  - In code, ensure `$djs_names` gathering covers all cases needed. The "fixed" code snippet you provided already streamlines this by dropping the unused `_event_djs` case. You can adopt that logic:
  - Loop through `_timetable` if available (and ensure `$timetable` is indeed an array; maybe your plugin saves it as serialized data – in the fixed code, they assume it's already an array via `get_post_meta(..., true)`).
  - If no DJs yet, check `_dj_name`.
  - Remove duplicates and empties, then format the string ($dj_display).
  - After implementing, test with an event where you manually set the DJ meta to verify the name appears. If still "Line-up em breve", it means the event has no data where the code expects; at that point, consider adjusting the plugin's event entry form or the code to use whatever field you have for DJ (for example, maybe the event content or excerpt was being used for DJs? If so, that's not captured in code – you could use `get_the_content()` as a last resort for DJ info if structured).

- **Multiple DJs and Display:** The code concatenates multiple DJ names and adds a "+X" if more than 3. Ensure this logic is preserved or works with your data. If your events typically have one DJ or a fixed set, this is fine. If the "+" is not desired, you could adjust the number or remove it for simplicity.

- **Event Location Not Showing:** If the location name or area isn't appearing, that means `_event_location` meta was empty or not in expected format for those events:

- The code expects `_event_location` to contain either `"Venue Name | Area"` (with a pipe separator) or just `"Venue Name"`. It then splits on `|` if present.
- Make sure your events have this meta. If not, perhaps the location was stored differently (maybe as a taxonomy or not at all). You have a few options:
  - Easiest is to edit each event and fill a custom field `_event_location` with something like `"Club XYZ | Downtown"` (or just `"Club XYZ"` if no area). Then the code will show "Club XYZ (Downtown)" or just "Club XYZ".
  - If you want to use a different source for location (like the event's post title or a taxonomy term), you'd need to modify the code. For example, if the event post's **category** or a specific taxonomy denotes location, you could fetch that instead. But since `_event_location` is clearly intended, using it is simplest.
- If location meta is set but still not showing, double-check the code path: it only prints location if `$event_location` is not empty. If the meta string was present but somehow not captured, debug the string. (For instance, a trailing space or a non-string value could cause an issue – but unlikely.)

- **Code Corrections:** The logic for location is straightforward and likely fine. Just ensure data is there. As a minor improvement, you could add a fallback: if `$event_location` is empty, maybe try to use the event's venue taxonomy if one exists (not seen in code, but some event plugins have a "venue" post type or taxonomy).

- **Other Variables:** Check if any other dynamic pieces are missing:

- Event date appears to be showing (the day and month were visible in the output).
- The event banner image is showing (the output had an `<img>` with src either from uploads or a placeholder).
- Category slug is used as a data attribute (for filtering). That is likely fine, but ensure each event had a category; otherwise it defaults to `'general'`. If you have an event with no category, the code uses 'general' slug — you might want to ensure all events are categorized or handle uncategorized differently if needed.
- The "Extra! Extra!" highlight banner is showing content from the latest blog post. In the output, it showed "Hello world!" post. If that's unintended (maybe you wanted a specific highlight post), you could adapt that query or select a specific post ID. But that's optional; the main issue is it works (it found the default post). If you prefer not to show the blog highlight at all, you could remove that section or query events instead of posts for a highlight event.

By correcting the data retrieval logic and ensuring the event posts have the necessary meta filled, the dynamic fields (DJ names, locations, etc.) should now display instead of default text. Test with a variety of events (with and without DJs, with and without location, multiple DJs, etc.) to confirm robustness.

C. Enabling the Lightbox Feature (JavaScript/CSS)

Currently, clicking on an event card likely does nothing (or just jumps to `#`) because the lightbox script isn't active. Here's how to fix the interactive modal that should open on card click:

- **Include the JS and CSS:** Make sure that the front-end includes the necessary JavaScript and CSS for the lightbox/modal:
- **JavaScript:** There is likely a script file (perhaps named `apollo-events-portal.js` or similar) that handles the click on `.event_listing` elements. This script probably:
    - Listens for clicks on event cards.
    - Fetches event details via AJAX (possibly by calling `apollo_modal_handler.php` or an AJAX endpoint that returns the event's full details like description, lineup, etc.).
    - Then populates the `#apollo-event-modal` container and displays it (likely by adding a class or manipulating `aria-hidden`).
    - It might also handle the layout toggle (switch between list and card view) and filtering by category/date.
- If this script isn't loaded, none of that happens. So, **enqueue the script** in the WordPress context. In the plugin's main PHP file or wherever appropriate (maybe on template load or via a shortcode callback), add something like:

```
wp_enqueue_script('apollo-events-portal-js', plugins_url('/assets/js/
apollo-events-portal.js', __FILE__), array('jquery'), '1.0', true);
```

and similarly enqueue a CSS if needed:

```
wp_enqueue_style('apollo-events-portal-css', plugins_url('/assets/css/
apollo-events-portal.css', __FILE__), array(), '1.0');
```

(Use the correct path/names for your files. The provided files `apollo_modal_css.css` and any relevant JS should be registered and enqueued.)

- Ensure the script is enqueued **only on the events portal page** (to avoid unnecessary loading elsewhere). You might use a condition or only enqueue when the shortcode is used or when `is_page('eventos')`.

- **Verify `toggleLayout` and Filter Functions:** The HTML has a layout toggle button (`#wpem-event-toggle-layout`) with an `onclick="toggleLayout(this)"`. This implies the JS defines a `toggleLayout()` function to switch between list and card views (probably adding/removing a CSS class or toggling `display` styles). Confirm that the JS file indeed provides this function and it's working:

- If not, you may need to implement it. For simplicity, if card view is desired by default, you might skip complex toggling and just ensure the CSS shows cards in a grid. However, it's nicer to have both views.

- Check if the **filter buttons** (the category buttons with `data-slug`) and the date picker (`#eventDatePicker`) require JS. Possibly the script filters the displayed events on the client side by hiding/showing based on data attributes, or it triggers new queries via AJAX. If those are meant to work, the script should handle their click events. Make sure the script covers this; if not, document that as a potential improvement (e.g., implement dynamic filtering via JS or AJAX in the future).

- **Modal Content Loading:** The `#apollo-event-modal` is an empty container for the lightbox. The `apollo_modal_handler.php` (provided in files) likely outputs HTML for an event's detail (like full description, extended lineup, etc.) when called (maybe via an AJAX request passing `event_id`). Ensure that:

- The AJAX endpoint is properly set up (could be through WordPress AJAX or a custom endpoint). Possibly the script does a jQuery `$.get` or `$.ajax` to `apollo_modal_handler.php?event_id=123`. If so, that PHP should be included and accessible (maybe via an `admin-ajax.php` call or direct include).
- For now, test if clicking an event card does anything. If not, open the browser console to see if a JS error is thrown (likely `toggleLayout is not defined` or similar if script is missing).
- After enqueuing the script, test again: clicking an event card should populate the modal. If it still doesn't, you may need to debug the script:
    - Ensure the script is selecting the correct elements (maybe it expects a certain class or structure).
    - Check that the event IDs and other data attributes are correctly set (the template does set `data-event-id`, `data-category`, etc., which is good).
    - The script might use `data-event-id` to fetch more info. You might have to implement a WordPress AJAX action to return event details (title, full description, maybe images) and then display them. If not already present, you can create a simple AJAX handler in the plugin:

```
add_action('wp_ajax_nopriv_get_event_details',
'apollo_get_event_details');
add_action('wp_ajax_get_event_details',
'apollo_get_event_details');
function apollo_get_event_details() {
    $id = intval($_REQUEST['event_id']);
    // Query the event post by ID and output JSON or HTML
```

```
    // e.g., title, content, DJs list, etc.
    wp_send_json(...);
}
```

Then adjust the JS to call `admin-ajax.php?action=get_event_details&event_id=...`.

- However, if `apollo_modal_handler.php` is meant to be a standalone endpoint, ensure WordPress is loading it (perhaps via a custom rewrite or including it via admin-ajax as well). It might be simpler to integrate it into the normal WP AJAX mechanism as above.

- **CSS for Modal:** Use the CSS file (likely `apollo_modal_css.css`) to style the modal (position it, make it overlay, etc.). Enqueue it as mentioned. Confirm the CSS has rules for `#apollo-event-modal` and any contents (like making it visible when a certain class like `.open` is added). If not, you may need to add CSS to handle showing/hiding the modal on demand (e.g., `.apollo-event-modal { display: none; } .apollo-event-modal.open { display: block; }`, and maybe backdrop styling).

By properly including and configuring the JS/CSS, the event portal page will become interactive: clicking on an event card will open a modal with details, and the layout toggle and filters will function, greatly improving user experience.

D. Layout and Styling Adjustments

We need to ensure that the **event cards display in a responsive grid** (inline per row, wrapping as needed) and that there are no alignment issues:

- **Default to Card Layout:** If currently all event cards are stacked vertically (as in the output, each anchor had `style="display: block;"`), it looks like the page loaded in "list view". We probably want a card grid by default. You can change the initial state:
- In the HTML, the layout toggle button's `aria-pressed` attribute might control default state. In the provided PHP, it was set to `aria-pressed="true"` with a list icon (meaning list view active). In the output, we saw `aria-pressed="false"` and a different icon (card icon), possibly after toggling or due to JS.
- To start in card view, set `aria-pressed="false"` in the HTML and perhaps change the icon accordingly (ri-building-3-fill was shown for cards). However, the better approach is to let the CSS and JS handle it. Possibly on page load the JS might toggle it off if `aria-pressed` is true.
- Simpler: remove any inline `style="display:block"` on the event anchors in the HTML template. Those might have been inserted by JS for list layout. Ensure that by default (without JS interference) the CSS for `.event_listing` allows them to sit inline (likely they should be inline-block or flex items).
- If the CSS class `discover-events-now-shortcode` or others were supposed to apply a grid style, check your CSS file. Perhaps `.event_listings` or `.event_listing` classes have styling. If not, we can add custom CSS:

```
.event_listings {
  display: flex;
  flex-wrap: wrap;
  gap: 20px; /* some spacing between cards */
  align-items: flex-start;
}
```

```
.event_listing {
  display: inline-block; /* each card as an inline-block or flex item */
  width: 300px; /* or any appropriate card width */
  vertical-align: top;
  /* additional styling like margin-bottom if needed */
}
```

The above ensures multiple `.event_listing` anchors will appear side by side until the row is full, then wrap to next line, keeping alignment at the top of each row.

- Alternatively, a CSS grid could be used:

```
.event_listings {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
  gap: 20px;
}
```

This automatically creates as many columns as fit in the container width.

- **Fix Misalignment:** If some cards were taller (due to different title lengths, etc.) and causing layout shifts, using flex or grid as above with `align-items: flex-start` will keep the tops aligned. Also ensure images have a max-width: 100% and maybe a consistent height or aspect ratio to avoid one very tall image breaking the row height. If needed, set a fixed height for the image container `.picture` or use CSS object-fit to crop uniformly.

- **Responsive Behavior:** Check that on smaller screens the cards stack or scroll nicely. The grid approach with `minmax` helps with responsiveness. Also consider that the filter bar and other elements should wrap or scroll on overflow (the provided code already uses a horizontal scroll for filters if too many, presumably via CSS).

- **CSS Cleanup:** Ensure that any modifications don't break existing styles:

- The template likely relies on an icon library (the `ri-*` classes, presumably RemixIcon), ensure that's loaded (maybe via a CDN or already in theme).
- Check the color scheme (dark mode toggle at bottom) – make sure the modal and new layout styles also look good in dark mode if that's supported. The dark mode toggle likely adds a class to `<body>` or toggles some CSS variables; make sure our added CSS works in both modes (e.g., use currentColor or not hard-code black text if body might be dark).

After adjusting the layout CSS and default states, the event cards should appear in neat rows. Test by resizing the browser to see how it wraps, and toggle the "list/cards" view if that functionality is implemented. In card view, all cards should be the same width and flow nicely; in list view (if toggled), each event might become full-width (the JS could simply add an inline style or class to set them block at 100%). Confirm the toggle button updates its icon and state accordingly.

With these corrections in place (query fixes, data output, JS/CSS, and layout), reload the portal page. You should now see: - Real event data populating (titles, DJs, locations, dates, images). - Events listed in an attractive grid by default. - The filter buttons (Todos/House/PsyTrance/etc.) filtering the visible events,

and the date picker adjusting which events show (if the JS for these is functional). - Clicking on an event opens a modal with more details. - Overall, a fully functional “Eventos” portal page.

Future Enhancements and Scalability

Finally, consider ways to **expand and improve the system** for both end users and administrators. Here are some suggestions:

- **Modularize and Clean Code:** Refactor the code into reusable components. For example, separate the data-fetching logic from the HTML template. You could have a function (or class method) that returns upcoming events data (an array of events with all needed fields), and then the template simply loops over that. This would make it easier to maintain and to reuse the logic (say, if you want the events list to appear in other places or in an API).

- Consider using WordPress’s Template Hierarchy or a templating engine for parts of the card HTML if this grows. For instance, each event card could be its own small template file to include, making it simpler to adjust card layout in one place.

- **Admin Interface Improvements:** Make it easier for administrators to input all relevant data:

- Ensure the custom post type **Event** has fields for all needed info (date, location, DJs, banner). Using Advanced Custom Fields (ACF) or similar could provide a user-friendly interface to fill these, if not already.
- If DJs are a key part of events, you might create a **DJ custom post type**. Then in events, instead of free-text for DJ names, the admin can select linked DJ posts. This way, you can reuse DJ profiles across events, and even have a DJs directory. The current code partly supports this (by checking if DJ is numeric and then looking up a post). To fully utilize it, implement a relationship field (maybe store selected DJ IDs in `_event_djs` meta). Then you can eliminate the “Line-up em breve” as each event would have at least something.
- Provide an admin page or menu for the Apollo Events Manager plugin for configuration. For example, options to set default images, toggle caching, or define categories.

- **Event Submission/Editing:** If the site expects user-generated events or many events, consider integrating a form for adding events from the front-end or improving the backend UI with custom columns (like showing start date, etc., in the admin list view for events).

- **Enhanced User Experience:** Add features to engage users:

- **Search and Filter:** The interface already has a search box and filter buttons. Expand on these by implementing the functionality:
    - Make the search box actually filter events by keywords (e.g., event title, DJ name, location). This can be done client-side (filter the already loaded events in the DOM) for simplicity if the list isn’t huge, or via AJAX querying the server for matching events.
    - The category buttons (House, Techno, etc.) currently likely filter by data-category. Ensure the JS actively filters the `.event_listing` elements when a category button is clicked (show/hide events based on their `data-category` attribute). Also update the active button styling accordingly.
    - The date picker (with prev/next month buttons and a display) suggests the ability to jump through months of events. Currently, all upcoming events are shown. In future, you might implement month-by-month filtering:

    - e.g., the JS could maintain a current month state and filter the events list to those in that month (the template includes `data-month-str` on each event, like "nov" for November). So clicking next could hide events not in the next month. This would require all upcoming events to be loaded or an AJAX call to get events of a specific month if not preloaded.
    - These interactive filters improve usability when there are many events.
- **Event Details Page or Modal:** If the modal is implemented, ensure it shows comprehensive info (date/time, full description, lineup, venue address, perhaps a map). You might add more to the modal content (like a link to a dedicated event page, social share buttons, etc.). If users might want a separate page for events (for linking or SEO), ensure each event has its own URL (the custom post type archive or single pages) and maybe link the modal's "Saiba Mais" button or the event title to that page.
- **Performance Considerations:** As the number of events grows, loading all upcoming events on one page could get heavy. You might:
    - Implement **pagination or lazy loading** for events. For example, load the next month's events only when the user clicks "next month" (via AJAX).
    - Use caching wisely: the transient per day is one way. You could also cache rendered HTML fragments or use a CDN if this becomes publicly large.
    - If many users use it concurrently, ensure the AJAX endpoints are efficient (they mostly read data, which is fine).

- **Accessibility & UX:** Ensure the modal is accessible (focus trapping, keyboard close on ESC, etc.). Also, the dark mode toggle indicates the site supports dark mode – verify the event portal looks good in both light and dark themes (text contrast, background of cards, etc.). If needed, adjust CSS or inherit theme styles for dark mode.

- **Scalability & Maintenance:**

- As more features are added, maintain a clear structure: e.g., have a dedicated JavaScript file for the portal interactions, a CSS file for styling, and keep the PHP focused on data. This separation makes future upgrades (like switching to a JS framework or using the WordPress REST API) easier.
- Consider implementing **REST API endpoints** for events and using a JS framework (React/Vue) for a more dynamic single-page application feel. This might be overkill now, but could be a future direction if real-time updates or complex state (bookmarks, user logins affecting content) are needed.

- Keep an eye on WordPress updates or plugin conflicts. For example, ensure compatibility with the active theme (the header and footer usage is good) and avoid relying on specific theme styles too heavily. If the plugin is used on another site/theme, you want it to still function (so include all necessary CSS within the plugin or clearly document theme requirements).

- **Additional Features:** To enhance user engagement and admin control, think about:

- **Calendar Integration:** Allow users to add events to Google Calendar or iCal easily (provide an "Add to Calendar" link with appropriate ICS file or link).
- **RSVP/Tickets:** If applicable, integrate a way for users to RSVP or buy tickets for events. This could be a simple link or a more involved integration with an event ticketing system.
- **Notifications:** If users can sign up, perhaps allow them to "follow" certain event types or DJs and get notified of new events (could be a longer-term feature using email or push notifications).

- **Admin Dashboard Widgets:** Show upcoming events or recent signups in the WP Admin dashboard for a quick overview.
- **Modular Extensions:** Design the plugin so that new features (like the ones above) can be added as modules or add-ons. For example, a separate module for ticketing, one for notifications, etc., to keep the core lightweight.

By implementing the fixes and considering these enhancements, the Apollo Events Manager portal will not only function correctly (displaying dynamic data and interactive elements as intended) but will also be well-positioned for future growth, easier maintenance, and a better experience for both users and administrators.

Next Step: Please proceed to apply the above debugging steps and code corrections. Begin by analyzing the provided code versus output, then implement the necessary changes in the PHP template and any related files (JavaScript, CSS). After making changes, test the page thoroughly to confirm that dynamic content is loading and the layout/interactive features work as expected.
