Technical Report: Plan Widget Session Mechanism Analysis

Executive Summary

This report provides a comprehensive analysis of the Plan Widget session mechanism within the **politeia-bookshelf** WordPress plugin. The system manages reading plans with scheduled sessions, providing users with calendar-based interfaces to create, move, and delete reading sessions.

---

1. Calendar Implementations

1.1 Reading Planner Form Calendar

Location: reading-plan.js

Purpose: Used during plan creation via the `[politeia_reading_plan]` shortcode. Allows users to select session dates when setting up a new reading plan.

Key Characteristics:

- Renders in Step 3 (Strategy) of the plan creation wizard
- Calculates sessions based on `pages_per_session` and `sessions_per_week` strategy
- Sessions are computed client-side and submitted to backend on plan acceptance
- Supports two goal types: `complete_books` and `form_habit`
- For `complete_books` : distributes total pages across planned session dates
- For `form_habit` : creates 48 consecutive daily sessions with progressive intensity

State Management:

```
state = {
  formData: {
    goals: [],
    baselines: {},
    books: [],
    startPage: 1,
    pages_per_session: null,
    sessions_per_week: null,
  },
  calculatedPlan: { pps: 0, durationWeeks: 0, sessions: [], type: 'ccl' },
  currentMonthOffset: 0,
  currentViewMode: 'calendar',
}
```

1.2 Plan Widget Calendar (My Plans Page)

Location: my-plans.php

Purpose: Displays existing plans on the `/members/{username}/my-plans/` page. Shows session status (planned, accomplished, missed) and allows session manipulation.

Key Characteristics:

- Renders via PHP template with inline JavaScript
- Data passed via `data-*` attributes on `.prs-plan-card` elements
- Supports **drag-and-drop** session rescheduling
- Supports **adding** sessions by clicking empty cells
- Supports **removing** sessions via remove button
- Visual indicators: gold (planned), black (accomplished), gray (missed)
- Navigation between months with prev/next buttons
- Dual view modes: Calendar and List

Calendar Rendering Logic (Lines 1871-2057):

```
const renderCalendar = () => {
  // Builds month grid
  // Applies status classes: is-missed, is-accomplished
  // Drag-drop handlers for session movement
  // Add/remove buttons for session management
};
```

1.3 Behavioral Differences

Feature	Reading Planner Form	Plan Widget
Purpose	Plan creation	Plan management
Persistence	Temporary (until submit)	Direct state display
Drag-drop	Not applicable	✓ Supported (frontend only)
Add/Remove	Part of calculation	✓ Supported (frontend only)
Status display	None	✓ Planned/Accomplished/Missed
Month navigation	Based on projected dates	✓ Full navigation
Backend sync	On plan submission	⚠️ **Changes not persisted**

> *[!IMPORTANT] The Plan Widget calendar **does not currently persist** session changes (add, remove, move) to the database. These changes are purely visual and are lost on page refresh. This is a significant implementation gap.*

---

2. Frontend → Backend Communication

2.1 Plan Creation Flow

Endpoint: `POST /wp-json/politeia/v1/reading-plan`

Handler: Rest::accept_plan()

Request Payload:

```
{
  "name": "Book Title",
  "plan_type": "custom|ccl",
  "pages_per_session": 20,
  "sessions_per_week": 3,
  "goals": [{
    "goal_kind": "complete_books|habit",
    "metric": "pages_total|pages_per_session|daily_threshold",
    "target_value": 300,
    "period": "plan|session|day",
    "book_id": 123,
    "starting_page": 1
  }],
  "baselines": { ... },
  "planned_sessions": [{
    "planned_start_datetime": "2026-01-30 00:00:00",
    "planned_end_datetime": "2026-01-30 23:59:59",
    "planned_start_page": 1,
    "planned_end_page": 20,
    "expected_number_of_pages": 20
  }]
}
```

Processing:

1. Validates authentication and nonce
2. Validates strategy parameters for `complete_books` plans
3. **Starts database transaction**
4. Inserts plan into `wp_politeia_plans`
5. Inserts goals into `wp_politeia_plan_goals`
6. Inserts sessions into `wp_politeia_planned_sessions`
7. **Commits transaction**
8. Creates user baseline record

2.2 Session Recording Flow

Endpoint: `wp_ajax_prs_save_reading`

Handler: Politeia_Reading_Sessions::ajax_save()

Flow:

1. User records a reading session via Session Recorder
2. Session saved to `wp_politeia_reading_sessions`
3. Automatically calls `mark_planned_session_accomplished()` (Line 266)
4. Updates corresponding planned session status to `'accomplished'`

2.3 Session Status Updates (My Plans Template)

Location: my-plans.php Lines 1008-1130

Inline Updates (on page load):

1. **Mark Missed Sessions:**

```
$wpdb->query($wpdb->prepare(
    "UPDATE {$sessions_table}
    SET status = 'missed'
    WHERE plan_id = %d
    AND status = 'planned'
    AND DATE(planned_start_datetime) < %s",
    $plan_id,
    $today_key
));
```

2. **Mark Accomplished Sessions:**

```
$wpdb->query($wpdb->prepare(
    "UPDATE {$sessions_table}
    SET status = 'accomplished'
    WHERE plan_id = %d
    AND status != 'accomplished'
    AND DATE(planned_start_datetime) IN ({$placeholders})",
    array_merge(array($plan_id), $actual_session_dates)
));
```

2.4 Desist Plan Flow

Endpoint: `wp_ajax_desist_reading_plan`

Handler: prs_handle_desist_plan()

Action: Updates plan status from `'accepted'` to `'desisted'`

---

3. Database Schema

3.1 Table Definitions

Schema Location: class-installer.php

---

wp_politeia_plans

Column	Type	Description
`id`	BIGINT UNSIGNED	Primary key
`user_id`	BIGINT UNSIGNED	Owner user ID
`name`	VARCHAR(255)	Plan name (usually book title)
`plan_type`	VARCHAR(50)	`custom`, `ccl`, etc.
`status`	VARCHAR(50)	`accepted`, `desisted`, `completed`
`pages_per_session`	INT UNSIGNED	Strategy: pages per session

`sessions_per_week`	INT UNSIGNED	Strategy: sessions per week
`created_at`	DATETIME	Creation timestamp
`updated_at`	DATETIME	Last update timestamp

---

wp_politeia_plan_goals

Column	Type	Description
`id`	BIGINT UNSIGNED	Primary key
`plan_id`	BIGINT UNSIGNED	FK to plans
`goal_kind`	VARCHAR(50)	`complete_books`, `habit`
`metric`	VARCHAR(50)	`pages_total`, `pages_per_session`, `daily_threshold`
`target_value`	INT UNSIGNED	Target pages to read in plan
`period`	VARCHAR(50)	`plan`, `session`, `day`
`book_id`	BIGINT UNSIGNED	FK to books (nullable)
`subject_id`	BIGINT UNSIGNED	FK to subjects (nullable)
`starting_page`	INT UNSIGNED	Page to start reading from (default: 1)

---

wp_politeia_planned_sessions

Column	Type	Description
`id`	BIGINT UNSIGNED	Primary key
`plan_id`	BIGINT UNSIGNED	FK to plans
`planned_start_datetime`	DATETIME	Session start (set to midnight)
`planned_end_datetime`	DATETIME	Session end (set to 23:59:59)
`planned_start_page`	INT UNSIGNED	Expected start page
`planned_end_page`	INT UNSIGNED	Expected end page
`expected_number_of_pages`	INT UNSIGNED	Pages to read in session
`expected_duration_minutes`	INT UNSIGNED	Expected duration
`status`	VARCHAR(50)	`planned`, `accomplished`, `missed`
`previous_session_id`	BIGINT UNSIGNED	For rescheduled sessions
`comment`	TEXT	Optional notes

---

wp_politeia_plan_subjects / wp_politeia_plan_participants

These tables support multi-subject plans and shared/group reading plans (currently unused in the analyzed flows).

3.2 Related Tables (Reading Module)

Table	Purpose
`wp_politeia_reading_sessions`	Actual reading sessions recorded by users
`wp_politeia_books`	Canonical book records
`wp_politeia_user_books`	User's book library with page counts

---

4. Session Recalculation Logic

4.1 Frontend Recalculation (Plan Widget)

Function: `buildDerivedPlan()` (my-plans.php Lines 1763-1842)

Triggered When:

- Calendar is rendered
- Sessions are added, removed, or moved
- View switches between calendar and list

Algorithm:

1. Calculate total pages read from actual sessions
2. Filter remaining planned sessions (status = 'planned', date >= today)
3. Distribute remaining pages evenly across remaining sessions
4. Apply extra pages to earlier sessions if not evenly divisible

```
const buildDerivedPlan = () => {
  const completedPages = Math.min(totalPages, actualPages);
  const remainingPages = Math.max(0, totalPages - completedPages);
  const remainingCount = remainingDates.length;
  const basePages = remainingCount > 0 ? Math.floor(remainingPages / remainingCount)
: 0;
  const extraPages = remainingCount > 0 ? remainingPages % remainingCount : 0;

  remainingDates.forEach((dateStr, index) => {
    const pages = basePages + (index < extraPages ? 1 : 0);
    // Assign start/end pages to session
  });
};
```

4.2 Backend Recalculation

> *[!WARNING]* ***There is no backend recalculation mechanism.*** *Page ranges stored at plan creation are never updated. The frontend* `buildDerivedPlan()` *recalculates displayed values, but these are not persisted.*

4.3 Status Transitions

```
stateDiagram-v2
    [*] --> planned: Session created
    planned --> accomplished: User records reading session on that date
    planned --> missed: Date passes without reading session
    missed --> missed: Cannot change (final state)
    accomplished --> accomplished: Cannot change (final state)
```

Transition Rules:

- **planned → accomplished:** Triggered by `mark_planned_session_accomplished()` when actual reading session date matches planned date
- **planned → missed:** Triggered on page load in my-plans.php when `DATE(planned_start_datetime) < today AND status = 'planned'`

---

5. Progress Calculation

Location: my-plans.php Lines 1195-1223

```php
// Get the highest page read
$highest_page_read = $wpdb->get_var(
    "SELECT MAX(end_page) FROM {$reading_sessions_table}
    WHERE user_id = %d AND book_id = %d AND deleted_at IS NULL"
);

// Calculate pages read within plan range
if ($highest_page_read >= $goal_starting_page) {
    $pages_read_in_plan = $highest_page_read - $goal_starting_page + 1;
} else {
    $pages_read_in_plan = 0;
}

// Progress percentage
$progress = min(100, floor(($pages_read_in_plan / $plan_target_pages) * 100));
```

---

6. Identified Issues and Recommendations

6.1 Critical: Session Changes Not Persisted

Issue: Session add/remove/move operations in the Plan Widget are frontend-only. Changes are lost on page refresh.

Recommendation: Implement AJAX endpoints:

- `POST /wp-json/politeia/v1/reading-plan/{plan_id}/session` - Create session
- `PUT /wp-json/politeia/v1/reading-plan/session/{id}` - Update session
- `DELETE /wp-json/politeia/v1/reading-plan/session/{id}` - Delete session

6.2 Inconsistency: Page Range Updates

Issue: When sessions are added or removed, page ranges should be recalculated and persisted to maintain accurate plan data.

Recommendation: Add backend logic to recalculate and update `planned_start_page` and `planned_end_page` for all remaining planned sessions when any session changes.

6.3 Edge Case: Missed Session Handling on Plan Widget Load

Issue: Missed sessions are marked on every page load. If the user didn't truly miss a session (e.g., read later than planned), there's no way to correct this.

Recommendation: Consider adding an "undo missed" functionality or grace period before marking as missed.

6.4 Session Time Handling

Issue: Sessions are stored with full datetime but normalized to midnight/end-of-day. The actual time component is not used.

Current Behavior:

```
$planned_start = $start_dt->setTime(0, 0, 0)->format('Y-m-d H:i:s');
$planned_end = $start_dt->setTime(23, 59, 59)->format('Y-m-d H:i:s');
```

---

7. Data Flow Diagram

```
flowchart TB
    subgraph Frontend
        A[Reading Planner Form] -->|Submit| B[REST API]
        C[Plan Widget Calendar] -->|Display| D[Session Data]
        C -->|Drag/Drop| E[Local State Only]
        F[Session Recorder] -->|Save| G[AJAX Handler]
    end

    subgraph Backend
        B -->|Transaction| H[(wp_politeia_plans)]
        B -->|Transaction| I[(wp_politeia_plan_goals)]
        B -->|Transaction| J[(wp_politeia_planned_sessions)]
        G -->|Insert| K[(wp_politeia_reading_sessions)]
        G -->|Update| J
    end

    subgraph Status_Updates
        L[Page Load] -->|Mark Missed| J
        L -->|Mark Accomplished| J
    end
```

---

8. File Reference Index

File	Purpose	Key Functions/Classes
class-rest.php	REST API endpoints	`accept_plan()`, `session_recorder()`
reading-plan.js	Form calendar & session calculation	`renderStepStrategy()`, `submitPlan()`
my-plans.php	Plan Widget template	`renderCalendar()`, `buildDerivedPlan()`
class-reading-sessions.php	Reading session AJAX handlers	`ajax_save()`, `mark_planned_session_accomplished()`
class-installer.php	Database schema	`get_schema_sql()`, `install()`
shortcodes.php	Shortcode registration	`render_reading_plan_shortcode()`

---

9. Conclusion

The Plan Widget session mechanism is a comprehensive system for managing reading plans, but has a significant gap: **session modifications in the Plan Widget are not persisted to the database**. The underlying architecture is sound, with proper transaction handling for plan creation and automatic status updates based on reading activity. Implementing the missing persistence layer for session CRUD operations would complete the system and provide full parity between the two calendar interfaces.