Root cause analysis

Finanzas SD – Architecture, Flows & SOPs

Arquitectura, Flujos y Procedimientos

December 14, 2025

1 Root cause analysis

- **Primary root cause:** The `/projects` handler scans DynamoDB with a hard `Limit` (default 100) and returns only the first page, ignoring `LastEvaluatedKey`. The frontend calls `/projects?limit=100` once and never paginates. When the projects table exceeds 100 records, end-user-created projects beyond the first page are omitted from the API response.
- **Evidence:** Handler code before fix used a single `ScanCommand` with `Limit: limit` and no `ExclusiveStartKey` loop (`services/finanzas-api/src/handlers/projects.ts`). The API client requests `/projects?limit=100` (`src/api/finanzas.ts`) and normalizes the first payload only. Seed/demo projects remain visible because they occupy early records in the scan; user-generated rows are pushed to later pages and dropped.
- **Secondary contributors:** Recent seed removal increased reliance on live data, raising the project count past 100 and exposing the pagination gap. # Root Cause Analysis: SDMT Data Flow Regression

Date: 2025-12-13
Issue: UI showing only subset of projects (end-user created projects missing)
Status: □ ROOT CAUSE IDENTIFIED

1.1 Primary Root Cause (CONFIRMED)

Inconsistent payload normalization in frontend project loading

1.1.1 The Problem

The main project loading path (`src/lib/api.ts::ApiService.getProjects()`) uses **incomplete** payload extraction logic that doesn't handle all the response shapes that the backend API can return.

1.1.2 Data Flow Chain

```
UI Component (ProjectContext)
  ↓
  calls ApiService.getProjects()
  ↓
  calls this.request(API_ENDPOINTS.projects) → GET /projects?limit=100
  ↓
  Backend handler (services/finanzas-api/src/handlers/projects.ts)
  ↓
  DynamoDB Scan with FilterExpression (sk = METADATA OR META)
```

```
↓
Maps to ProjectDTO via mapToProjectDTO()
↓
Returns: { data: [...projects], total: N }
↓
Response travels back through API Gateway / CloudFront
↓
Frontend receives payload
↓
**BREAK POINT:** ApiService.getProjects() extraction logic
↓
Tries to extract array from payload
↓
Limited pattern matching (only 4 patterns checked)
↓
Falls through to empty array []
↓
ProjectContext receives empty/partial list
↓
UI shows empty or incomplete project dropdown
```

1.1.3 Where Data Disappears

Location: `src/lib/api.ts` lines 156-164

```
[] const projectArray = Array.isArray(payload) ? payload // Pattern 1: Direct array
: Array.isArray(payload?.data) ? payload.data // Pattern 2: { data: [...] } ✅ SHOULD
WORK : Array.isArray(payload?.items) ? payload.items // Pattern 3: { items: [...] }
: Array.isArray(payload?.data?.items) ? payload.data.items // Pattern 4: { data: {
items: [...] } } : []; // ❌ FALLBACK: Empty array!
```

Expected Backend Response: `{ data: [...], total: N }`
Should Match: Pattern 2 (`payload.data`) ✅
So why do projects disappear?

1.1.4 The Missing Link

PR #606 introduced `normalizeProjectsPayload()` helper that checks **13+ patterns**:
1. `payload.data` 2. `payload.items` 3. `payload.data.items` 4. `payload.projects` ✅
5. `payload.Items` ✅ (DynamoDB style) 6. `payload.results` ✅ 7. `payload.records`

□ 8. `payload.body` □ 9. `payload.body.items` □ 10. `payload.body.Items` □ 11. `payload.body.results` □ 12. `payload.body.records` □ 13. And nested variants...

But `ApiService.getProjects()` only checks patterns 1-4!

1.1.5 What Exactly Breaks

Scenario A: CloudFront/API Gateway Transformation

```
[] // Backend sends: { "data": [...], "total": 10 }
// CloudFront/API Gateway wraps it: { "body": { "data": [...], "total": 10 } }
// ApiService extraction: □ payload (not array) □ payload.data → undefined (it's at payload.body.data!) □ payload.items → undefined □ payload.data.items → undefined □ Falls through to []
```

Scenario B: Environment-Specific Format

```
[] // Dev/Stage environment (direct API): { "data": [...], "total": 10 } □ Works
// Prod environment (via API Gateway): { "Items": [...] } □ Doesn't work (pattern not checked)
```

Scenario C: Backend Update

```
[] // If backend handler changes to return: { "projects": [...], "count": 10 }
// ApiService extraction: □ Fails (payload.projects not checked)
```

1.2 Evidence Supporting Root Cause

1.2.1 1. Code Structure Analysis

PR #606 introduced: - □ `normalizeProjectsPayload()` in `src/api/finanzas-projects-helper` - □ Used in `src/modules/finanzas/projects/useProjects.ts` - □ **NOT** used in `src/lib/api.ts` (critical path)

ProjectContext usage:

```
[] // src/contexts/ProjectContext.tsx line 166 const projectData = await ApiService.getProjects(); // ← Uses ApiService
```

ApiService implementation:

[] // src/lib/api.ts line 150 **static async** getProjects(): Promise<Project[]> { **const** payload = **await this**.request(API_ENDPOINTS.projects); // ⚠ Uses limited inline extraction, NOT normalizeProjectsPayload() **const** projectArray = Array.isArray(payload) ? payload : ...

1.2.2 2. Test Coverage Gap

Tests for `normalizeProjectsPayload()`: - ✅ 7 test cases covering all patterns - ✅ Validates extraction from various shapes

Tests for `ApiService.getProjects()`: - ❌ No tests verifying it handles alternate shapes - ❌ No tests comparing it with normalizeProjectsPayload()

1.2.3 3. Backend Response Verification

Current backend handler (line 1253 in projects.ts):

[] **return** ok({ data: projects, total: projects.length });

This returns `{ data: [...] }` which **SHOULD** work with Pattern 2.

But: - CloudFront might wrap it - API Gateway might transform it - Different environments might have different configurations - Future backend changes might alter format

1.2.4 4. DynamoDB Query Correctness

Backend query (lines 1178-1193):

[] FilterExpression: "begins_with(#pk, :pkPrefix) AND (#sk = :metadata OR #sk = :meta)", ExpressionAttributeValues: { ":pkPrefix": "PROJECT#", ":metadata": "METADATA", ":meta": "META", }

This correctly filters for: - ✅ All projects with pk starting with "PROJECT#" - ✅ Both METADATA and META sort keys (backward compat)

Seed projects: Created with `sk = "METADATA"` ✅
User projects: Created with `sk = "METADATA"` (line 1076) ✅

Conclusion: DynamoDB query is NOT the problem.

1.3 Secondary Contributors

1.3.1 1. Pagination Limit (Minor)

Limit: 100 projects per request (line 1166)

```
[] const limit = Math.min(Math.max(requestedLimit, 1), 100);
```

No pagination tokens returned: Backend doesn't return `nextToken` or `lastEvaluatedKey`

Impact: - If >100 projects exist, only first 100 returned - Frontend has no way to request more - Could cause "some projects missing" symptom

Mitigation: Not the primary issue (most deployments have <100 projects)

1.3.2 2. RBAC Filtering (Minor)

Roles that see all projects: - ADMIN - EXEC_RO - PMO - SDMT

Roles with limited access: - SDM: Only projects where `sdm_manager_email` matches user email

Potential issue: - If user role not correctly identified - Or if `sdm_manager_email` not set on projects - User might see empty list

Mitigation: Not the primary issue (symptom appears for ADMIN users too)

1.3.3 3. Environment/Table Configuration (Minor)

Potential mismatch: - Frontend `VITE_API_BASE_URL` points to wrong environment - Backend `TABLE_PROJECTS` env var points to wrong DynamoDB table - CloudFront distribution routes to wrong API

Mitigation: Would affect ALL projects, not just end-user ones

1.4 Why Seeded/Demo Projects Appear But End-User Projects Don't

Theory 1: Different Code Paths - Seed projects might be loaded via a different path that uses `normalizeProjectsPayload()` - End-user projects loaded via main path (ApiService) that doesn't

Theory 2: Response Format Differences - Seed data endpoints return `{ data: [...] }` □ - User data endpoints return `{ Items: [...] }` □ - Only seed format matches ApiService extraction patterns

Theory 3: Pagination - Seed projects alphabetically first (e.g., "P-SEED-*") - Fit within limit=100 - User projects alphabetically last (e.g., "P-USER-*") - Truncated by limit

Theory 4: Caching - Seed projects cached in browser/CloudFront - User projects not cached (recently created) - Stale cache returns old format that works

Most Likely: Theory 1 or Theory 2 (format mismatch)

1.5 Proof of Root Cause

1.5.1 Test to Confirm

1. Add logging to `ApiService.getProjects()`:

```
[] logger.info("Raw API response:", payload); logger.info("Extracted project array:", projectArray);
```

2. Compare with backend logs:

```
[] console.info("[projects] Returning projects", { count: projects.length, format: "{ data: [...], total: N }" });
```

3. If logs show:

   - Backend: "Returning 50 projects"
   - Frontend: "Extracted 10 projects" or "Extracted 0 projects"
   - **ROOT CAUSE CONFIRMED**

1.5.2 Expected Outcome

After applying fix (using `normalizeProjectsPayload()`): - Backend: "Returning 50 projects" - Frontend: "Extracted 50 projects" - □ All projects visible in UI

1.6 The Fix

File: `src/lib/api.ts`
Line: 152
Change: Use canonical `normalizeProjectsPayload()` helper

```
[] import { normalizeProjectsPayload } from "@/api/finanzas-projects-helpers";
// Inside ApiService class static async getProjects(): Promise<Project[]> { try { const payload = await this.request(API_ENDPOINTS.projects);
logger.info("Projects loaded from API:", payload);
// Use canonical payload normalization const projectArray = normalizeProjectsPayload(payload);
if (!Array.isArray(projectArray)) { throw new Error("Invalid projects payload from API"); }
// ... rest of function unchanged
```

Why This Works: 1. Handles all 13+ response patterns consistently 2. Matches behavior of `useProjects` hook 3. Makes code resilient to backend format changes 4. Well-tested (7 test cases) 5. Minimal change (2 lines)

1.7 Risk Assessment

Change Risk: LOW - Single function change - Import existing well-tested helper - No logic changes beyond extraction - Backward compatible (handles current format + more)

Rollback: EASY - Revert single commit - No database changes - No API contract changes

Testing: STRAIGHTFORWARD - Verify project dropdown loads - Check all projects visible - Test with different user roles - Confirm SDMT views work

---

CONCLUSION:

The root cause is **incomplete payload normalization** in `src/lib/api.ts::ApiService.getPr` The function doesn't use the canonical `normalizeProjectsPayload()` helper introduced in PR #606, causing it to fail when API responses use alternate formats. Fix: Import and use `normalizeProjectsPayload()` at line 152. This is a **2-line change** with **high confidence** and **low risk**.