

# **Finanzas Write Operations Wiring - Implementation Summary**

Finanzas SD - Architecture, Flows & SOPs  
Arquitectura, Flujos y Procedimientos

December 17, 2025

# 1 Finanzas Write Operations Wiring - Implementation Summary

**Date:** 2025-11-13

**Branch:** copilot/wire-main-actions-to-api

**Status:** ☐ Complete

## 1.1 Overview

This implementation wires the Finanzas UI to the backend API endpoints for all write operations (create/update) as specified in the OpenAPI spec. The solution ensures proper error handling, JWT validation, and user-friendly feedback for all scenarios.

## 1.2 Changes Made

### 1.2.1 1. Enhanced finanzasClient API Layer

**File:** src/api/finanzasClient.ts

#### New Write Methods (5 total):

- `createProject(payload) → POST /projects`
- `createProjectRubro(projectId, payload) → POST /projects/{projectId}/rubros`
- `saveAllocations(projectId, payload) → PUT /projects/{projectId}/allocations:bulk`
- `createAdjustment(payload) → POST /adjustments`
- `createProvider(payload) → POST /providers`

#### TypeScript Schemas Added:

- `ProjectCreateSchema / ProjectSchema`
- `RubroCreateSchema`
- `AllocationBulkSchema`
- `AdjustmentCreateSchema`
- `ProviderCreateSchema`

#### Enhanced Error Handling:

- **501 (Not Implemented):** "This feature is not yet implemented on the server (501). The backend handler needs to be completed."
- **401 (Unauthorized):** "You must be signed in to perform this action. Please log in with your Finance credentials."
- **403 (Forbidden):** "You must be signed in and have the Finance role to perform this action. Please check your permissions."
- **HTML Response Detection:** Detects when API returns HTML instead of JSON (wrong URL or login page redirect)

## **JWT Validation:**

- `checkAuth()` function validates JWT presence before write operations
- Checks `cv.jwt`, `finz_jwt`, or `STATIC_TEST_TOKEN`
- Throws descriptive error if no token found

### **1.2.2 2. New UI Components**

#### **ProjectsManager (src/modules/finanzas/ProjectsManager.tsx)**

- Full project creation form with validation
- Fields: name, code, client, start/end dates, budget, currency, description
- Code format validation: PROJ-YYYY-NNN
- Wired to `finanzasClient.createProject()`
- Comprehensive error handling and toast notifications

#### **AdjustmentsManager (src/modules/finanzas/AdjustmentsManager.tsx)**

- Budget adjustment creation with distribution options
- Fields: project ID, type, amount, date, distribution method, justification
- Supports: exceso, reduccion, reasignacion
- Distribution methods: `pro_rata_forward`, `pro_rata_all`, `single_month`
- Wired to `finanzasClient.createAdjustment()`

#### **ProvidersManager (src/modules/finanzas/ProvidersManager.tsx)**

- Provider/vendor registration
- Fields: name, tax ID, type, contact info, country, notes
- Provider types: servicios, materiales, software, infraestructura
- Wired to `finanzasClient.createProvider()`

### **1.2.3 3. Enhanced Existing Components**

#### **RubrosCatalog (src/modules/finanzas/RubrosCatalog.tsx)**

- Added “Agregar a Proyecto” button for each rubro
- Dialog form with fields:
  - Project ID (with format hint)
  - Monto Total (numeric)
  - Tipo de Ejecución (mensual, puntual, por\_hito)
  - Notas (optional)
- Wired to `finanzasClient.createProjectRubro()`
- Full error handling with specific messages for each scenario

**FinanzasHome (src/modules/finanzas/FinanzasHome.tsx)**

- Added navigation cards for all new modules:
  - Proyectos
  - Catálogo de Rubros
  - Reglas de Asignación
  - Ajustes Presupuestarios
  - Proveedores

**1.2.4 4. Routing Updates****File:** src/App.tsx

Added new routes (feature-flagged with VITE\_FINZ\_ENABLED): - /projects → ProjectsManager - /adjustments → AdjustmentsManager - /providers → ProvidersManager

Updated imports for new components.

**1.3 Error Handling Patterns**

All components implement consistent error handling:

```
[try { setIsSubmitting(true); const result = await finanzasClient.createXXX(payload);
toast.success("Success message"); setDialogOpen(false); // Reset form } catch (e:
any) { const errorMessage = e?.message || "Default error message";
  if (errorMessage.includes("501")) { toast.error("This feature is not yet implemented
on the server (501)..."); } else if (errorMessage.includes("signed in") || errorMes-
sage.includes("401") || errorMessage.includes("403")) { toast.error(errorMessage);
} else if (errorMessage.includes("HTML") || errorMessage.includes("login page")) {
toast.error("Could not connect to Finanzas API. Please contact support."); console.error("API
configuration issue:", errorMessage); } else { toast.error(errorMessage); } } finally
{ setIsSubmitting(false); }
```

**1.4 Network Behavior****1.4.1 Expected Network Requests**

All write operations will trigger the following HTTP requests:

**1. Create Project**

- Method: POST
- URL: {VITE\_API\_BASE\_URL}/projects

- Headers: Authorization: Bearer {jwt}, Content-Type: application/json
- Body: JSON with project data

## 2. Add Rubro to Project

- Method: POST
- URL: {VITE\_API\_BASE\_URL}/projects/{projectId}/rubros
- Headers: Same as above
- Body: JSON with rubro data

## 3. Save Allocations

- Method: PUT
- URL: {VITE\_API\_BASE\_URL}/projects/{projectId}/allocations:bulk
- Headers: Same as above
- Body: JSON with allocations array

## 4. Create Adjustment

- Method: POST
- URL: {VITE\_API\_BASE\_URL}/adjustments
- Headers: Same as above
- Body: JSON with adjustment data

## 5. Create Provider

- Method: POST
- URL: {VITE\_API\_BASE\_URL}/providers
- Headers: Same as above
- Body: JSON with provider data

### 1.4.2 Response Handling

- **200/201 Success:** Display success toast, close dialog, reset form
- **401/403 Auth Error:** Show “must be signed in” message
- **404 Not Found:** Show “resource not found” message
- **501 Not Implemented:** Show “feature not yet implemented (501)” message
- **HTML Response:** Show “API misconfigured” message with console error
- **Other Errors:** Show error message from API or generic fallback

## 1.5 Testing Checklist

### 1.5.1 Local Development Testing

□ **Build:** All components build successfully without errors

□ **Manual Testing (requires API):** - [ ] Create Project with valid data → Network tab shows POST /projects - [ ] Create Project without JWT → Shows auth error - [ ] Add Rubro to Project → Network tab shows POST /projects/{id}/rubros - [ ] Save Allocations → Network tab shows PUT /projects/{id}/allocations:bulk - [ ] Create Adjustment → Network tab shows POST /adjustments - [ ] Create Provider → Network tab shows POST /providers - [ ] Test with 501 backend response → Shows “not implemented” message - [ ] Test with HTML response → Shows “API misconfigured” message - [ ] Verify data persists after successful create operations

### 1.5.2 Production Testing

□ **Deployment Testing (requires deploy):** - [ ] Verify at <https://d7t9x3j66yd8k.cloudfront.net> - [ ] Test all write operations - [ ] Verify Network tab behavior - [ ] Confirm error handling for various scenarios

## 1.6 Security

□ **CodeQL Analysis:** No security issues found

### 1.6.1 Security Measures Implemented:

1. JWT validation before all write operations
2. No secrets committed to code
3. Input validation with TypeScript schemas
4. XSS protection via React’s built-in escaping
5. CORS credentials set to “omit”
6. No inline scripts or dangerous HTML rendering

## 1.7 GREEN Criteria Status

□ **Every key “write” action sends HTTP request to API** - All 5 write operations properly wired

□ **No write button silently “does nothing”** - All buttons show success toast or error message - Clear “Not implemented (501)” notice for stub endpoints

□ **No changes to infra or SAM templates** - All changes in src/ directory only - No modifications to infrastructure

□ **User-friendly error messages** - 501: Clear “not implemented” message - 401/403: Auth guidance - HTML: API configuration issue - Generic: Fallback error message

□ **JWT guard for write operations** - All write methods call checkAuth() - Clear error if JWT missing

## 1.8 Dependencies

No new dependencies added. Using existing: - sonner for toast notifications - zod for schema validation - lucide-react for icons - @/components/ui/\* for UI components

## 1.9 File Structure

```
src/
├── api/
│   └── finanzasClient.ts           (Extended with write methods)
├── modules/finanzas/
│   ├── FinanzasHome.tsx          (Updated navigation)
│   ├── RubrosCatalog.tsx        (Enhanced with add to project)
│   ├── ProjectsManager.tsx      (NEW)
│   ├── AdjustmentsManager.tsx   (NEW)
│   └── ProvidersManager.tsx      (NEW)
└── App.tsx                       (Added routes)
```

## 1.10 Next Steps

1. **Manual Testing:** Test with actual Finanzas API when deployed
2. **Backend Implementation:** Complete handler stubs that return 501
3. **Data Persistence:** Verify data saves to DynamoDB and appears on refresh
4. **Documentation:** Update FINANZAS\_AUTH\_IMPLEMENTATION\_SUMMARY.md if needed

## 1.11 Notes

- AllocationRulesPreview is intentionally read-only (no write operations)
- SDMT Cost/Catalog uses separate ApiService (mock), not finanzasClient
- All forms include validation matching OpenAPI spec requirements
- Build passes successfully with no TypeScript errors
- All components follow consistent patterns for maintainability