

Finanzas SD - Invoice Reconciliation

Finanzas SD - Architecture, Flows & SOPs

Arquitectura, Flujos y Procedimientos

December 11, 2025

1 Finanzas SD – Invoice Reconciliation

1.1 1. Overview

Purpose.

Enable Ikusi finance / SDMT teams to match real vendor invoices against forecast allocations (line items + months) to validate spend, track variances, and close the loop on the R1 Execution Plan item **“Finanzas SD – Reconciliation.”**

End-to-end flow (current implementation).

1. User selects a **project** in the header.
2. In **Reconciliation**, user opens “Upload Invoice,” fills invoice metadata, and attaches a file.
3. UI calls the **presign upload** endpoint to obtain an S3 uploadUrl + objectKey and records metadata in the docs table.
4. Browser performs a direct **PUT** to the shared documents bucket using the pre-signed URL.
5. UI calls **POST /projects/{projectId}/invoices** with invoice details + S3 documentKey.
6. Backend creates an INVOICE record linked to the project/line item and marks status as Pending.
7. Reconciliation tiles (Matched / Pending / Disputed) and the invoice grid **refetch and refresh**.

1.2 2. Data Model

1.2.1 2.1 INVOICE entity (data dictionary vs current state)

Target data-dictionary design

- invoice_id – **PK**, pattern inv_[a-z0-9]{10}, **system-generated**.
- project_id – **FK** → **PROJECT.project_id**, required.
- invoice_number – business invoice number from the vendor, **user-entered**.
- currency – ISO-4217, **user-entered**.
- amount – numeric invoice total, **user-entered**, > 0.
- invoice_date – YYYY-MM-DD, **user-entered**.
- forecast_id – **optional FK** → **FORECAST_ALLOCATION.forecast_id** (links to the specific allocation).

Current implementation

- invoice_id

- Generated server-side. Today uses an INV-... style identifier (e.g. INV-<short-uuid>).
- **Action item:** migrate to the data-dictionary pattern `inv_[a-z0-9]{10}` when stable.
- `project_id`
 - Required; must match the `{projectId}` path parameter.
 - UI enforces project selection via `ProjectContext`.
- `invoice_number`
 - Required in the UI; stored as `invoice_number` (vendor reference).
 - In earlier versions the backend could fall back to INV-<timestamp> if missing; UI now always supplies it, so fallback can be removed or left as defensive behavior.
- `amount`
 - Stored as numeric; UI validates `> 0`.
- `invoice_date`
 - Accepted as YYYY-MM-DD; stored as ISO timestamp (2025-11-01T00:00:00.000Z) after parsing.
- `currency`
 - **Not yet persisted**; UI formats with a default currency (usually USD or project currency).
 - **Action item:** add currency to the table + payload.
- `forecast_id`
 - **Not yet persisted.** Matching today is based on `line_item_id` + month.
 - **Action item:** introduce `forecast_id` once FORECAST_ALLOCATION is finalized.

1.2.2 2.2 Related entities

- **PROJECT**
 - Reconciliation is scoped to a single project (`selectedProjectId`).

- In Dynamo, invoices live under a project-centric partition (pk pattern PROJECT#<projectId> or equivalent), in the **invoices/prefacturas table** (exact table name is environment-specific).
- **FORECAST_ALLOCATION** (plan vs real)
 - For each (project_id, rubro_id, month) there is (or will be) a forecast allocation row.
 - **Current matching:** invoice links to a line item and a numeric month; variance is computed on the fly by joining invoices with forecast data.
 - **Target:** add a direct forecast_id FK.
- **PROVIDER**
 - UI captures vendor as **free text** on the invoice.
 - A PROVIDER / providers table already exists for the broader platform.
 - **Action item:** decide whether to map vendor to a provider ID or keep free text for R1.
- **Document metadata**
 - Each upload gets a documentId and metadata row in a **docs table** (bucket, key, module, project, uploader, timestamps, warnings).
 - Reconciliation invoices reference those documents by S3 objectKey.

1.2.3 2.3 Generated vs user-entered identifiers

- **System-generated**
 - invoice_id (currently INV-..., target inv_[a-z0-9]{10}).
 - documentId / S3 objectKey.
 - Timestamps (uploaded_at, updated_at).
 - uploaded_by (Cognito subject / email).
- **User-entered**
 - invoice_number.
 - vendor.
 - amount.
 - invoice_date.
 - description (if present).

- line_item_id (selected from catalog).
- month.
- Original file name + content type (derived from the upload).

1.3 3. S3 Document Handling

1.3.1 3.1 Bucket & key layout

- **Bucket:**
ukusi-ui-finanzas-prod
(injected into the upload Lambda via DOCS_BUCKET / DocsBucketName).
- **Key structure (current):**

```
[ ] docs/{projectId}/{module}/{contextPart}-{random}-{safeFilename}
```

Where:

- module = "reconciliation" for this flow (other modules: prefactura, catalog, changes).
- contextPart prefers lineItemId, else invoiceNumber, else a timestamp.
- safeFilename is the original file name normalized by replacing disallowed chars (e.g., [^A-Za-z0-9._-] → _).

1.3.2 3.2 Flow

1. UI calls **POST /uploads/docs** with:

```
[ ] { "projectId": "PROJ-123", "module": "reconciliation", "lineItemId": "RUBRO-1", "invoiceNumber": "FAC-9981", "invoiceDate": "2025-11-01", "vendor": "Ikusi Vendor", "amount": 12500, "contentType": "application/pdf", "originalName": "invoice-nov.pdf" }
```

2. Upload Lambda (upload-docs.ts):

- Validates required fields per module (projectId, module, basic file metadata; for reconciliation also lineItemId, amount > 0, valid invoiceDate, vendor).
- Computes objectKey under docs/{projectId}/reconciliation/....
- Persists a metadata record in the docs table (project, module, documentId, objectKey, etc.).

- Returns:

```
[ ] { "uploadUrl": "https://ukusi-ui-finanzas-prod.s3.us-east-2.amazonaws.com/...",
      "objectKey": "docs/PROJ-123/reconciliation/RUBRO-1-ab12-invoice-nov.pdf",
      "documentId": "DOC-ab12", "bucket": "ukusi-ui-finanzas-prod", "metadata":
      { "...": "..." }, "warnings": [ ] }
```

3. Browser performs a **direct PUT** to S3:

```
[ ] PUT {uploadUrl} Content-Type: <file.type>
<file bytes>
```

4. On success, the UI then calls POST /projects/{projectId}/invoices (see below) with the documentKey and other invoice details.

1.3.3 3.3 CORS requirements (confirmed)

On ukusi-ui-finanzas-prod we configured:

```
[ ] [ { "AllowedOrigins": [ "https://d7t9x3j66yd8k.cloudfront.net" ], "AllowedMethods": [ "GET", "PUT", "POST", "HEAD" ], "AllowedHeaders": [ "*" ], "ExposeHeaders": [ "ETag" ], "MaxAgeSeconds": 300 } ]
```

This is the **minimal, working configuration**:

- Allows the Finanzas CloudFront origin to PUT objects.
- Lets the browser read basic headers (ETag).
- Has been validated end-to-end by successfully uploading invoices from the Reconciliation screen.

1.4 4. API Contracts

1.4.1 4.1 Presign upload (documents)

Method / URL

- POST /uploads/docs

Request (Reconciliation)

```
[ ] { "projectId": "PROJ-123", "module": "reconciliation", "lineItemId": "RUBRO-1",
"invoiceNumber": "FAC-9981", "invoiceDate": "2025-11-01", "vendor": "Ikusi Ven-
dor", "amount": 12500, "contentType": "application/pdf", "originalName": "invoice-
nov.pdf" }
```

Response 201

```
[ ] { "uploadUrl": "https://ukusi-ui-finanzas-prod.s3.us-east-2.amazonaws.com/...",
"objectKey": "docs/PROJ-123/reconciliation/RUBRO-1-ab12-invoice-nov.pdf", "documen-
tId": "DOC-ab12", "bucket": "ukusi-ui-finanzas-prod", "metadata": { "...": "..." },
"warnings": [ ] }
```

Validation & error behavior

- Validates projectId, module, contentType, originalName.
- For module = "reconciliation" also validates lineItemId, amount > 0, and invoiceDate.
- On failure:
 - 400 for missing/invalid fields.
 - 5xx for internal errors (S3, Dynamo, etc.), mapped via shared FinanzasApiError handler.

1.4.2 4.2 Create invoice (main reconciliation API)

Note: the create/update logic lives in the reconciliation backend (not in invoices.ts, which is a read-only alias). The UI interacts with it via POST /projects/{projectId}/invoices and status-update actions.

Method / URL

- POST /projects/{projectId}/invoices

Request body (current implementation)

```
[ ] { "projectId": "PROJ-123", "lineItemId": "RUBRO-1", "month": 11, "amount":
12500, "description": "Networking refresh", "vendor": "Ikusi Vendor", "invoiceNum-
ber": "FAC-9981", "invoiceDate": "2025-11-01", "documentKey": "docs/PROJ-123/reconciliation/RU
1-ab12-invoice-nov.pdf", "originalName": "invoice-nov.pdf", "contentType": "applica-
tion/pdf" }
```

Behavior

- Validates:
 - projectId in body matches {projectId} in path.
 - lineItemId belongs to the selected project / catalog.
 - month \in [1, 12].
 - amount > 0.
 - invoiceDate parses to a valid date (if present).
- Generates invoiceId (currently INV-...) and sets:
 - status = "Pending".
 - uploaded_by from Cognito user.
 - uploaded_at timestamp.
- Persists invoice in the invoices/prefacturas table under the project partition.

Response 201

```
[ ] { "id": "INV-ab12", "project_id": "PROJ-123", "line_item_id": "RUBRO-1", "month": 11, "amount": 12500, "status": "Pending", "vendor": "Ikusi Vendor", "invoice_number": "FAC-9981", "invoice_date": "2025-11-01T00:00:00.000Z", "documentKey": "docs/PROJ-123/reconciliation/RUBRO-1-ab12-invoice-nov.pdf", "originalName": "invoice-nov.pdf", "contentType": "application/pdf", "uploaded_by": "user@example.com", "uploaded_at": "2025-12-04T01:26:33.000Z" }
```

1.4.3 4.3 Status update (design target)

Note: as of this document, status updates are handled via the existing reconciliation backend endpoints; a dedicated REST endpoint is a **design goal**, not confirmed in production.

Proposed contract

- PUT /projects/{projectId}/invoices/{invoiceId}/status

Request:


```
[ ] { "status": "Matched", "comment": "Validated against November forecast" }
```

Behavior:

- Validate {projectId} and {invoiceId} exist and belong together.
- status ∈ { "Pending", "Matched", "Disputed" }.
- Persist new status + optional comment and updated_at.

1.4.4 4.4 GET /invoices alias (read-only)

Handler: **invoices.ts**

Method / URL

- GET /invoices?project_id={projectId}

Purpose

- Provide a **read-only alias** to fetch invoices/prefacturas for a project, primarily for frontend compatibility and legacy integrations.
- Internally, this queries the same underlying table used by the reconciliation module.

Behavior

- Requires project_id query string parameter; otherwise returns 400.
- Enforces read access via ensureCanRead.
- Queries the invoices/prefacturas table by project partition and returns:

```
[ ] { "data": [ /* invoice rows */, "projectId": "PROJ-123", "total": 3 }
```

- Returns 200 with an empty array if no invoices are found.
- Returns 500 with a generic error message if DynamoDB or auth handling fails.

1.5 5. UI Flow (SDMT Reconciliation)

1. Select Project

- User chooses a project from the global project selector.
- All reconciliation queries use selectedProjectId.

2. Open Reconciliation

- Route: `/finanzas/sdmt/cost/reconciliation` (exact path per router config).
- Screen loads:
 - Summary tiles: Matched, Pending Review, Disputed Items.
 - Invoices grid for the selected project and period.

3. Upload Invoice

- Click **“Upload Invoice”**.
- Modal fields:
 - **Line Item** (maps to `lineItemId` from catalog).
 - **Month** (1-12).
 - **Invoice Amount** (amount).
 - **Vendor** (vendor).
 - **Invoice Number** (invoiceNumber – vendor reference).
 - **Invoice Date** (invoiceDate).
 - File input (file, with extension whitelist).
- On submit:
 1. Call **POST /uploads/docs** to get `uploadUrl` + `objectKey`.
 2. Perform **PUT to S3** with the selected file.
 3. Call **POST /projects/{projectId}/invoices** with invoice details + `documentKey`.

4. Success behavior

- Toast: “Invoice and document uploaded”.
- S3 now contains the file under the reconciliation prefix.
- Invoices grid refetches and shows the new row with status Pending.
- Summary tiles update counts for Pending Review.

5. Viewing & updating invoices

- Grid columns (current):
 - Line Item / Month.
 - Amount.
 - Status (badge).
 - Vendor.
 - Invoice Number.
 - File (filename with link to download via presigned GET).

- Users with appropriate permissions can:
 - Change status (e.g., Pending → Matched or Disputed).
 - Open related forecast / allocation screens (where configured).

6. Key UI clarifications

- **Invoice Number** (invoice_number) is **vendor-facing** and always user-editable in the form.
 - **Internal invoice ID** (id / invoice_id) is never edited; it's used for status updates and internal references.
 - The **currency shown** is currently implicit (e.g., USD). Once currency is persisted in the backend, the grid will format amounts using that value.
-

1.6 6. Status Logic

1.6.1 6.1 Current

- Supported statuses in code/UI:
 - Pending (default on creation).
 - Matched.
 - Disputed.
- Transitions are **explicit** (set by user action).
- Matching itself is **manual** (operator compares invoice vs forecast and chooses the status).
- Variance reporting and export are derived by joining invoices with allocations; no automatic “over/under” state is written back yet.

1.6.2 6.2 Planned extensions

- Add forecast_id to strongly tie an invoice to a specific allocation row.
 - Surface explicit variance metrics (amount, percentage) and color-coded thresholds.
 - Support bulk operations (e.g., approve all invoices in a filtered view).
-

1.7 7. Open Questions / TODOs

1. Identifier normalization

- Align `invoice_id` format to `inv_[a-z0-9]{10}` and update any consumers (Acta / Prefectura / reporting).

2. Currency handling

- Add currency column to the invoice table and wire it through the UI form + API.
- Decide default per project (e.g., from project metadata).

3. Forecast linkage

- Introduce `forecast_id` in the invoice schema.
- Update reconciliation UI to select a specific allocation where multiple exist for the same line item/month.

4. Provider linkage

- Decide whether vendor should remain free text or map to a PROVIDER record (with id, tax data, payment terms).

5. Validation / uniqueness

- Enforce uniqueness or at least soft checks on (`project_id`, `invoice_number`) to reduce duplicates.
- Add richer validation rules for `invoice_number` and `invoice_date` per Ikusi policy.

6. Search & filtering UX

- Add filters by status, vendor, date range, and amount range.
- Add pagination or virtual scrolling for projects with many invoices.

7. Document lifecycle

- Define retention and deletion rules for S3 objects and corresponding docs metadata:
 - When an invoice is deleted.
 - When an invoice is superseded (re-upload).
 - When a project is archived.

Quick note on the two backend files you just shared

- `**`upload-docs.ts`**` – matches this doc perfectly (modules set, presign, metadata)
- `**`invoices.ts`**` – is a `**GET-only alias**` (``/invoices?project_id=...``) that returns

When you're ready, we can move on to the rubros / line-items handlers and make sure