What the Full-Stack Cost Plans System Is Intended to Do

The Full-Stack Cost Plans System is designed to serve as a **single, authoritative planning, analysis, and decision-support platform** for managing cost plans across projects, countries, organizational units, and planning years. Its primary purpose is to replace fragmented spreadsheet and no-code workflows with a structured, governed system that supports accurate planning, controlled flexibility, and reliable reporting.

At its core, the system enables finance users to **plan costs within a clear project-year context**. Every interaction begins with selecting a planning year and a project, ensuring that all data entry, calculations, and analysis are scoped correctly. Projects act as the primary working context, while planning years define the temporal boundary for assumptions, parameters, and outputs.

The system supports two major classes of planning inputs: **staff-related costs** and **non-staff operating and programme costs**. For staffing, users plan positions, grades, duty stations, and deployment over time. These inputs are combined with centrally managed salary and cost parameters to derive accurate staff cost calculations. For non-staff costs, users plan operating and programme cost items using structured classifications, quantities, rates, and time distributions.

A key function of the system is the **strict separation between input data, calculation rules, and derived outputs**. Users never edit calculated values directly. Instead, all totals, quarterly values, and summaries are produced through deterministic computation logic that can be rerun safely and consistently. This ensures transparency, traceability, and predictable recomputation when assumptions change.

The system is explicitly built to support **scenario-based planning and what-if analysis**. A baseline plan always exists for each project and year, representing the official cost plan. Finance users can create named scenarios to test alternative assumptions, cost reductions, reallocations, or policy changes without altering the baseline. Scenarios apply overrides in a controlled layer, allowing direct comparison of baseline versus scenario outcomes and clear identification of cost impacts and savings.

Reporting and dashboards are a core output of the system. Users can view costs aggregated at multiple organizational levels, including project, section or country, branch, division, and organization-wide. Reports support breakdowns by cost category, Umoja class, quarter, and year. The system enables baseline versus scenario comparisons, year-on-year analysis, and savings measurement, making cost plans an active decision-support tool rather than a static reporting artifact.

Governance and auditability are first-class concerns. The system records who changed what, when, and why, maintains versioned snapshots of plans, and tracks calculation runs. This ensures that cost plans remain explainable, reviewable, and suitable for audit and management review.

In summary, this full-stack project exists to **turn cost planning into a structured, reliable, and analyzable process**, enabling finance teams to plan, compare, justify, and adjust costs with confidence, while providing management with clear, trustworthy insight for financial decision-making.

Full-Stack Cost Plans System FlowT

1) Planning cycle setup (Year is the boss)

1. Admin creates/opens a **Planning Year** (example: 2026).
2. System marks it as **Draft / Active / Closed**.
3. If closed, system locks editing for that year (audit-safe).

2) Project onboarding into a planning year

1. Projects already exist in **projects (WBSE master)**.
2. For a selected year, create **project_metadata (project-year context)**:
    a. Division, Branch, Section/Country
    b. Trust Fund, Grant, Earmarking, etc.
3. Assign finance ownership using **user_project_access** (My Projects).

3) Auto-initialize the project cost plan (templates)

1. Finance clicks: **"Initialize Cost Plan for Project-Year"**
2. System auto-creates baseline rows:
    a. OPC items from opc_templates
    b. Deployment scaffolding (by grade/category or by position)
    c. Optional: activity scaffolding
3. Everything starts consistent, not "Jane's Excel version 4."

4) Reference + Parameters selection

1. Select or attach the correct **salary_parameter_set** for:
    a. Year + duty station/country (and later: version)
2. Set **project_parameters**:
    a. Inflation
    b. PSC rate (Umoja 155)
    c. Defaults like currency assumptions if needed

5) Staffing planning (position-first, not vibes-first)

1. System maintains allowed positions via **project_positions**.
2. Finance builds a **staffing_plan** per project-year:
    a. Position ID, category, grade, duty station
    b. Funding dates or coverage period
3. Later (integration): system syncs position details from **Umoja** into the position layer.

6) Deployment allocation (time dimension)

1. Finance sets allocations via:
    a. Monthly allocations, or
    b. Date range + percent rules (system expands into months)
2. System calculates **post-months** per position or per grade bucket.

3. Deployments can be split across projects where needed.

7) Non-staff costs entry (OPC)

1. Finance edits **opc_items**:
    a. Quantity, unit cost, duration
    b. Umoja class, commitment item, object code
2. Allocate across quarters (Q1–Q4) or let system derive.
3. Items can be toggled for scenarios (what-if).

8) Activities and results framework (optional but supported)

1. Define **activities** (project-year).
2. Add **activity_costs**:
    a. Quarterly allocations
    b. Classifications for reporting
3. Enables reporting by “what we’re doing”, not just “what we’re spending”.

9) Scenario planning (baseline vs what-if)

1. Baseline scenario always exists.
2. Finance creates new scenarios:
    a. “No Petrol”
    b. “Cut travel by 30%”
    c. “Shift budget to staffing”
3. Scenario changes stored as **scenario_overrides** (not destructive edits).
4. System can compare scenarios side-by-side.

10) Versioning and approvals (immutability)

1. Finance works in **Draft**.
2. When ready: create a **Version**:
    a. Submitted
    b. Approved
    c. Released snapshot
3. Approved versions become immutable.
4. Audit trail stays clean and boring (which is the goal).

11) Recalculation engine (real logic, not spreadsheet praying)

1. Finance triggers **Recompute** (or system runs automatically on save later).
2. System runs deterministic calculations:
    a. staff_costing
    b. psc_calculations
    c. summary_lines
    d. rollup_totals
3. Every run is logged in **calculation_runs** with status + timestamp.

12) Reporting and Dashboards (multi-level + What-if Scenarios)

12.1 Year-first reporting (mandatory filter)

1. User selects **Planning Year** (ex: 2026).
2. System loads reporting context for that year only:
   a. Baseline + available scenarios
   b. Latest approved version (or draft if allowed)

12.2 Multi-level rollups (same numbers, different lenses)

Users can view totals and breakdowns at:

- **Project (WBSE)**
- **Section/Country**
- **Branch**
- **Division**
- **Organization-wide**

And slice by:

- **Umoja Class** (010, 155, etc.)
- **Category / Subcategory**
- **Commitment item / IMIS object code**
- **Cost type (staff vs non-staff)**
- **Quarter (Q1–Q4) and Annual**

12.3 Baseline vs Scenario comparison (core decision-support)

1. Baseline always exists (the official plan).
2. Users can select **1 or more scenarios** and compare:
   a. Baseline vs Scenario A
   b. Baseline vs Scenario A vs Scenario B
3. System shows deltas:
   a. **Absolute difference (CHF/USD)**
   b. **% change**
   c. **Which category changed most**
   d. **Which org level benefits most (division/branch/country)**

12.4 What-if scenario types supported

Finance users can create scenarios like:

A) Cost item toggles (simple switches)

- Exclude or reduce specific OPC items (ex: **petrol**, **travel**, **IT equipment**)
- Apply at:
  - project level

- country level
- division level

B) Category-level adjustments (bulk policy changes)

- “Cut all travel by 20% across Division X”
- “Reduce operating costs by 10% in Country Y”
- “Freeze recruitment for 6 months” (staffing impact)

C) Reallocation scenarios (trade-off planning)

- “Remove petrol, redirect savings to 2 new posts”
- “Shift budget from consultants to staffing”
- “Move activity funding from Q4 to Q2”

D) Parameter-based what-if

- Alternate inflation assumptions
- Different PSC rates (if policy changes)
- Alternate exchange rate assumptions (if included)

12.5 Scenario drill-down and explainability

For any scenario delta, the system must let the user drill into:

- **Which override caused it**
- The affected records (OPC items / staffing / deployment / activities)
- The override reason/comment (mandatory for governance)

So the dashboard isn’t just “numbers changed”, it’s “numbers changed because *this* changed.”

12.6 YoY comparison with scenarios (the “savings proof” view)

Users can compare:

- **2025 baseline vs 2026 baseline**
- **2026 baseline vs 2026 scenario (expected savings)**
- **2025 baseline vs 2026 scenario (policy impact across years)**

This enables reporting like:

- “We reduced petrol spend by X from last year”
- “Scenario A saves Y across Division 1”
- “Category Z increased despite overall savings”

12.7 Outputs and integration

Reporting must support:

- Dashboard tables + charts (web UI)
- Export:
  - CSV / Excel
  - Power BI-friendly dataset
- Scheduled extracts (later phase)

13) Security & access (so nobody edits random projects)

1. Finance users see **My Projects** by default.
2. Reviewers get read-only or approval permissions.
3. Admins manage years, templates, reference data, roles.

14) Audit + governance (so this survives UN reality)

1. Every change is logged in **audit_log**.
2. Imports logged in **data_import_jobs**.
3. Calculation runs logged in **calculation_runs**.
4. You can always answer: *who changed what, when, and why*.

3. High-Level System Architecture

3.1 Architecture at a Glance

The Cost Plans system will be a **year-scoped, project-centric** full-stack application that supports:

- **Structured planning inputs** (staffing, deployment, OPC, activities)
- **Scenario-based overrides** (what-if analysis without touching baseline)
- **Deterministic computation** (recompute derived outputs on demand)
- **Multi-level reporting** (project → country/section → branch → division → org-wide)

Core architectural principle: **separate inputs, rules, and outputs**, with strong traceability and auditability.

Layers

- Frontend (role-based UI + navigation + scenario tools)
- Backend API (workflow + validation + permissions)
- Calculation Engine (recompute + rule evaluation + explainability)
- Database (authoritative source of truth + versioning)
- Reporting layer (dashboards + exports + aggregates)

3.2 Frontend Overview

3.2.1 Primary User Experiences

The UI is designed around two dominant workflows:

1. **Finance Users ("My Projects")**
    a. Default landing: assigned projects for the selected planning year
    b. Guided flows for entering/updating:
        i. Staffing plan
        ii. Deployment allocations
        iii. OPC items
        iv. Activities and activity costs
    c. Scenario creation and toggles (baseline vs what-if)
    d. Trigger recompute and view "calculation status"
2. **Reviewers / Management (Exploration + Reporting)**
    a. Browse all projects within a selected year
    b. Filter by metadata: division, branch, country/section, trust fund, grant, etc.
    c. Roll-up views and comparisons
    d. Read-only access (or controlled commentary/approval workflows)

3.2.2 Navigation Model

- **Year first** (mandatory selection)
- Then hierarchical drill-down:
    - Year → Division → Branch → Section/Country → Project (WBSE)
- Plus faceted filtering and grouping across metadata dimensions.

3.2.3 Key Screens

- Dashboard (year summary + major KPIs + trends)
- Project workspace (single project-year context)
- Scenario manager (create, compare, apply overrides)
- Recompute status + audit trail viewer
- Reporting hub (filters + exports + saved views)

3.3 Backend Services Overview

3.3.1 Responsibilities

Backend services must provide:

- Auth + role-based access control (RBAC)
- User-to-project authorization ("My Projects")
- CRUD APIs for planning inputs (staffing, deployment, OPC, activities)

- Scenario override APIs (non-destructive edits)
- Validation rules (data integrity, required relationships, allowed values)
- Recompute orchestration (trigger + status + results)
- Version lifecycle (draft → submitted → approved → frozen)

3.3.2 Suggested Service Split

This can be one service initially, but logically it breaks into:

- **Identity & Access Service**
  - Users, roles, user_project_access
- **Planning Data Service**
  - project_metadata + staffing + deployment + opc + activities
  - validations, templates
- **Scenario & Version Service**
  - scenarios, scenario_overrides, versions
- **Calculation Service**
  - compute staff_costing, PSC, summary_lines, rollups
  - store run history in calculation_runs
- **Reporting Service**
  - exports, prepared aggregations, dashboard endpoints

3.4 Database Layer Overview

3.4.1 Role of the Database

The database is the **authoritative source of truth** for:

- Planning inputs
- Reference data
- Scenarios and overrides
- Immutable versions/snapshots
- Derived outputs (computed tables)
- Audit and traceability

3.4.2 Design Expectations

- Planning year and project-year context are **first-class**
- Constraints enforce correctness (FKs, unique keys, not-null)
- Historical versions are **immutable**
- Derived tables are **recomputable** and traceable back to inputs
- Auditing is consistent across key transactional tables

3.5 Calculation and Rules Engine Overview

3.5.1 Why a Dedicated Engine Exists

This system cannot rely on “formulas everywhere” because:

- Exceptions are normal
- Scenario comparisons require parallel computation
- Outputs must be recomputed deterministically and explainably

So we implement a calculation engine that:

- Pulls baseline inputs
- Applies scenario overrides
- Applies rules in a defined precedence order
- Writes derived outputs into computed tables
- Logs the run, inputs hash, and status

3.5.2 Core Behaviors

- **Deterministic recompute**
    - same inputs → same outputs
- **Idempotent runs**
    - reruns don’t duplicate or corrupt results
- **Explainability hooks**
    - ability to trace how totals were produced
- **Incremental recompute (future enhancement)**
    - recompute only impacted parts instead of full project-year

3.5.3 Rule Application Order (Precedence)

Example precedence:

1. Global reference parameters (year + location)
2. Project-year assumptions (inflation, PSC, defaults)
3. Staffing plan / deployment allocations (inputs)
4. Scenario overrides (record-level changes)
5. Derived calculations (staff_costing → PSC → summary_lines → rollups)

3.6 Reporting and Analytics Layer

3.6.1 Reporting Goals

The system must support reporting at:

- Project (WBSE)
- Section/Country

- Branch
- Division
- Org-wide
- Plus by cost dimensions: category/subcategory, Umoja class, commitment item, etc.

3.6.2 What-If Scenarios in Reporting

Users must be able to:

- Compare **baseline vs scenario** side-by-side
- Run “remove petrol” or “reduce transport by 20%” at:
  - project level
  - country/section level
  - division level
- See impact on:
  - quarterly totals
  - annual totals
  - category breakdowns
  - PSC and grand totals

3.6.3 Outputs and Integrations

- Built-in dashboards (fast rollups)
- Export to CSV/Excel
- Power BI friendly outputs (clean summary_lines + rollup_totals)
- Saved filters/views for recurring reporting needs