CURRENT

PRESENT

ects Wiki

financial-planning-u / src / features / sdmt / cost / Forecast / SDMTForecast.tsx in main

Cancel changes | Commit changes...

Edit | Preview

@@ -1,126 +1,83 @@

```
1  import { useState, useEffect, useMemo, useRef, useCallback
   } from "react";
2  import { Card, CardContent, CardHeader, CardTitle } from
   "@/components/ui/card";
3  import { Button } from "@/components/ui/button";
4  import { Badge } from "@/components/ui/badge";
5  import { Input } from "@/components/ui/input";
6  import { Label } from "@/components/ui/label";
7  import {
8    Select,
9    SelectContent,
10   SelectItem,
11   SelectTrigger,
12   SelectValue,
13 } from "@/components/ui/select";
14 import {
15   Table,
16   TableBody,
17   TableCell,
18   TableHead,
19   TableHeader,
20   TableRow,
21 } from "@/components/ui/table";
22 import {
23   Tooltip,
24   TooltipContent,
25   TooltipProvider,
26   TooltipTrigger,
27 } from "@/components/ui/tooltip";
```

```
1  import { useState, useEffect, useMemo, useRef, useCallback } from 'react';
2  import { Card, CardContent, CardHeader, CardTitle } from
   '@/components/ui/card';
3  import { Button } from '@/components/ui/button';
4  import { Badge } from '@/components/ui/badge';
5  import { Input } from '@/components/ui/input';
6  import { Label } from '@/components/ui/label';
7  import {
8    Table,
9    TableBody,
10   TableCell,
11   TableHead,
12   TableHeader,
13   TableRow,
14 } from '@/components/ui/table';
15 import {
16   Tooltip,
17   TooltipContent,
18   TooltipProvider,
19   TooltipTrigger,
20 } from '@/components/ui/tooltip';
```

```
28    import {
29      Dialog,
30      DialogContent,
31      DialogDescription,
32      DialogHeader,
33      DialogTitle,
34      DialogTrigger,
35    } from "@/components/ui/dialog";
36    import {
37      Collapsible,
38      CollapsibleContent,
39      CollapsibleTrigger,
40    } from "@/components/ui/collapsible";
41    import {
42      Share2,
43      TrendingUp,
44      TrendingDown,
45      ExternalLink,
46      FileSpreadsheet,
47      Info,
48      Calculator,
49      ChevronDown,
50      Calendar,
51    } from "lucide-react";
52    import { toast } from "sonner";
53    import { ChartInsightsPanel } from
      "@/components/ChartInsightsPanel";
54    import LineChartComponent from
      "@/components/charts/LineChart";
55    import { StackedColumnsChart } from
      "@/components/charts/StackedColumnsChart";
56    import ModuleBadge from "@/components/ModuleBadge";
57    import LoadingSpinner from "@/components/LoadingSpinner";
58    import type { ForecastCell, LineItem } from
      "@/types/domain";
59    import { useAuth } from "@/hooks/useAuth";
60    import { ALL_PROJECTS_ID, useProject } from
      "@/contexts/ProjectContext";
```

```
21    import {
22      Dialog,
23      DialogContent,
24      DialogDescription,
25      DialogHeader,
26      DialogTitle,
27      DialogTrigger,
28    } from '@/components/ui/dialog';
29    import { Collapsible, CollapsibleContent, CollapsibleTrigger } from
      '@/components/ui/collapsible';
30    import { Share2, TrendingUp, TrendingDown, ExternalLink, FileSpreadsheet, Info,
      Calculator, ChevronDown } from 'lucide-react';
31    import { toast } from 'sonner';
32    import { ChartInsightsPanel } from '@/components/ChartInsightsPanel';
33    import LineChartComponent from '@/components/charts/LineChart';
34    import { StackedColumnsChart } from '@/components/charts/StackedColumnsChart';
35    import ModuleBadge from '@/components/ModuleBadge';
36    import LoadingSpinner from '@/components/LoadingSpinner';
37    import type { ForecastCell, LineItem } from '@/types/domain';
38    import { useAuth } from '@/hooks/useAuth';
39    import { ALL_PROJECTS_ID, useProject } from '@/contexts/ProjectContext';
40    import { handleFinanzasApiError } from
      '@/features/sdmt/cost/utils/errorHandling';
41    import { useNavigate, useLocation } from 'react-router-dom';
42    import { excelExporter, downloadExcelFile } from '@/lib/excel-export';
43    import { PDFExporter, formatReportCurrency, formatReportPercentage,
      getChangeType } from '@/lib/pdf-export';
44    import { computeTotals, computeVariance } from '@/lib/forecast/analytics';
45    import { normalizeForecastCells } from
      '@/features/sdmt/cost/utils/dataAdapters';
46    import { useProjectLineItems } from '@/hooks/useProjectLineItems';
47    import { bulkUploadPayrollActuals, type PayrollActualInput, getProjectRubros,
      getBaselineById, type BaselineDetail } from '@/api/finanzas';
48    import { getForecastPayload, getProjectInvoices } from './forecastService';
49    import finanzasClient, { type BaselineDetailResponse } from
      '@/api/finanzasClient';
50    import { ES_TEXTS } from '@/lib/i18n/es';
51    import { BaselineStatusPanel } from
      '@/components/baseline/BaselineStatusPanel';
52    import { BudgetSimulatorCard } from './BudgetSimulatorCard';
53    import { MonthlyBudgetCard } from './MonthlyBudgetCard';
```

```
61  import { handleFinanzasApiError } from "@/features/sdmt/cost/utils/errorHandling";
62  import { useNavigate, useLocation } from "react-router-dom";
63  import { excelExporter, downloadExcelFile } from "@/lib/excel-export";
64  import {
65    PDFExporter,
66    formatReportCurrency,
67    formatReportPercentage,
68    getChangeType,
69  } from "@/lib/pdf-export";
70  import { computeTotals, computeVariance } from "@/lib/forecast/analytics";
71  import { normalizeForecastCells } from "@/features/sdmt/cost/utils/dataAdapters";
72  import { useProjectLineItems } from "@/hooks/useProjectLineItems";
73  import {
74    bulkUploadPayrollActuals,
75    type PayrollActualInput,
76    getProjectRubros,
77    getBaselineById,
78    type BaselineDetail,
79  } from "@/api/finanzas";
80  import { getForecastPayload, getProjectInvoices } from "./forecastService";
81  import { normalizeInvoiceMonth } from "./useSDMTForecastData";
82  import finanzasClient from "@/api/finanzasClient";
83  import { ES_TEXTS } from "@/lib/i18n/es";
84  import { BaselineStatusPanel } from "@/components/baseline/BaselineStatusPanel";
85  import { BudgetSimulatorCard } from "./BudgetSimulatorCard";
86  import { MonthlyBudgetCard } from "./MonthlyBudgetCard";
87  import { PortfolioSummaryView } from "./PortfolioSummaryView";
88  import { ForecastSummaryBar } from "./components/ForecastSummaryBar";
89  import { ForecastChartsPanel } from "./components/ForecastChartsPanel";
90  import { ForecastRubrosTable } from "./components/ForecastRubrosTable";
```

```
54  import { PortfolioSummaryView } from './PortfolioSummaryView';
55  import { ForecastSummaryBar } from './components/ForecastSummaryBar';
56  import { ForecastChartsPanel } from './components/ForecastChartsPanel';
57  import { ForecastRubrosTable } from './components/ForecastRubrosTable';
58  import { TopVarianceProjectsTable } from './components/TopVarianceProjectsTable';
59  import { TopVarianceRubrosTable } from './components/TopVarianceRubrosTable';
60  import { MonthlySnapshotGrid } from './components/MonthlySnapshotGrid';
61  import { DataHealthPanel } from '@/components/finanzas/DataHealthPanel';
62  import type { BudgetSimulationState, SimulatedMetrics } from './budgetSimulation';
63  import { applyBudgetSimulation, applyBudgetToTrends } from './budgetSimulation';
64  import { isBudgetNotFoundError, resolveAnnualBudgetState } from './budgetState';
65  import {
66    allocateBudgetMonthly,
67    aggregateMonthlyTotals,
```

```
import { TopVarianceProjectsTable } from "./components/TopVarianceProjectsTable";
import { TopVarianceRubrosTable } from "./components/TopVarianceRubrosTable";
import { MonthlySnapshotGrid } from "./components/MonthlySnapshotGrid";
import { DataHealthPanel } from "@/components/finanzas/DataHealthPanel";
import type {
  BudgetSimulationState,
  SimulatedMetrics,
} from "./budgetSimulation";
import { applyBudgetSimulation, applyBudgetToTrends } from "./budgetSimulation";
import { isBudgetNotFoundError, resolveAnnualBudgetState } from "./budgetState";
import {
  allocateBudgetMonthly,
  aggregateMonthlyTotals,
  allocateBudgetWithMonthlyInputs,
  calculateRunwayMetrics,
  type MonthlyAllocation,
  type MonthlyBudgetInput,
  type RunwayMetrics,
} from "./budgetAllocation";
import {
  buildCategoryTotals,
  buildCategoryRubros,
  buildPortfolioTotals,
} from "./categoryGrouping";
import { buildProjectTotals, buildProjectRubros } from "./projectGrouping";
import { getTaxonomyById } from "@/lib/rubros/canonical-taxonomy";

import { getBaselineDuration, clampMonthIndex } from "./monthHelpers";

// ---- Exported helpers for month support and testing ----
// Re-exported from monthHelpers.ts for backward compatibility
export { getBaselineDuration, clampMonthIndex };
```

```
  allocateBudgetWithMonthlyInputs,
  calculateRunwayMetrics,
  type MonthlyAllocation,
  type MonthlyBudgetInput,
  type RunwayMetrics,
} from './budgetAllocation';
import {
  buildCategoryTotals,
  buildCategoryRubros,
  buildPortfolioTotals,
} from './categoryGrouping';
import { buildProjectTotals, buildProjectRubros } from './projectGrouping';
import { rubrosFromAllocations } from '@/features/sdmt/utils';
```

```
123  // ------------------------------------------------------
     --
124
125  // TODO: Backend Integration for Change Request Impact on
     Forecast
126  // When a change request is approved in SDMTChanges, the
     backend should:
```

```
81

82   // TODO: Backend Integration for Change Request Impact on Forecast
83   // When a change request is approved in SDMTChanges, the backend should:
```

```
•••  @@ -173,134 +130,63 @@ type LineItemLike = Record<string, unknown>;
```

```
173  // Constants
174  const MINIMUM_PROJECTS_FOR_PORTFOLIO = 2; // ALL_PROJECTS
     + at least one real project
175
```

```
130  // Constants
131  const MINIMUM_PROJECTS_FOR_PORTFOLIO = 2; // ALL_PROJECTS + at least one real
     project
132
```

```
176  // Feature flags for new forecast layout
177  const NEW_FORECAST_LAYOUT_ENABLED =
     import.meta.env.VITE_FINZ_NEW_FORECAST_LAYOUT === 'true';
178  const SHOW_KEY_TRENDS =
     import.meta.env.VITE_FINZ_SHOW_KEYTRENDS === 'true';
179
180  // Backward compatibility: HIDE_KEY_TRENDS is deprecated,
     use SHOW_KEY_TRENDS instead
181
182  // Feature flag to hide executive key-trends (projects &
     rubros) cards
183  const HIDE_KEY_TRENDS =
     import.meta.env.VITE_FINZ_HIDE_KEY_TRENDS === 'true';
184
185  // Feature flag to hide Real Annual Budget KPIs in
     TODOS/Portfolio view
186  const HIDE_REAL_ANNUAL_KPIS =
     import.meta.env.VITE_FINZ_HIDE_REAL_ANNUAL_KPIS ===
     'true';
187
188  // Feature flag to hide Resumen de Portafolio in TODOS
     mode
189  const HIDE_PROJECT_SUMMARY =
     import.meta.env.VITE_FINZ_HIDE_PROJECT_SUMMARY === 'true';
190
191  // Feature flag to show Portfolio KPI tiles in new layout
     (default false for minimal view)
192  const SHOW_PORTFOLIO_KPIS =
     import.meta.env.VITE_FINZ_SHOW_PORTFOLIO_KPIS === 'true';
193
194  // Debug logging for feature flags (development only)
195  if (import.meta.env.DEV) {
196    console.log('[SDMTForecast] Feature Flags:', {
197      NEW_FORECAST_LAYOUT_ENABLED,
```

```
198       SHOW_KEY_TRENDS,
199       HIDE_KEY_TRENDS,
200       HIDE_REAL_ANNUAL_KPIS,
201       HIDE_PROJECT_SUMMARY,
202       SHOW_PORTFOLIO_KPIS,
203     });
204   }
205
206   // Feature flags for portfolio summary view customization
207   // (Flags for ONLY_SHOW_MONTHLY_BREAKDOWN_TRANSPOSED, HIDE_EXPANDABLE_PROJECT_LIST,
208   // and HIDE_RUNWAY_METRICS are declared and used inside PortfolioSummaryView.tsx)
209
```

```
210   export function SDMTForecast() {
211     const [forecastData, setForecastData] = useState<ForecastRow[]>([]);
212     const [loading, setLoading] = useState(true);
213     const [isLoadingForecast, setIsLoadingForecast] = useState(true);
214     const [forecastError, setForecastError] = useState<string | null>(null);
215     const [exporting, setExporting] = useState<"excel" | "pdf" | null>(null);
216     const [editingCell, setEditingCell] = useState<{
217       line_item_id: string;
218       month: number;
219       type: "forecast" | "actual";
220     } | null>(null);
221     const [editValue, setEditValue] = useState("");
222     const [dataSource, setDataSource] = useState<"api" | "mock">("api");
223     const [generatedAt, setGeneratedAt] = useState<string | null>(null);
224     const [portfolioLineItems, setPortfolioLineItems] = useState<
225       ProjectLineItem[]
226     >([]);
227     const [dirtyActuals, setDirtyActuals] = useState<Record<string, ForecastRow>>(
228       {}
229     );
```

```
133   export function SDMTForecast() {
134     const [forecastData, setForecastData] = useState<ForecastRow[]>([]);
135     const [loading, setLoading] = useState(true);
136     const [isLoadingForecast, setIsLoadingForecast] = useState(true);
137     const [forecastError, setForecastError] = useState<string | null>(null);
138     const [exporting, setExporting] = useState<'excel' | 'pdf' | null>(null);
139     const [editingCell, setEditingCell] = useState<{ line_item_id: string; month: number; type: 'forecast' | 'actual' } | null>(null);
140     const [editValue, setEditValue] = useState('');
141     const [dataSource, setDataSource] = useState<'api' | 'mock'>('api');
142     const [generatedAt, setGeneratedAt] = useState<string | null>(null);
143     const [portfolioLineItems, setPortfolioLineItems] = useState<ProjectLineItem[]>([]);
144     const [dirtyActuals, setDirtyActuals] = useState<Record<string, ForecastRow>>({});
```

```
230     const [savingActuals, setSavingActuals] = useState(false);
231     const [dirtyForecasts, setDirtyForecasts] = useState<
232       Record<string, ForecastRow>
233     >({});
234     const [savingForecasts, setSavingForecasts] = useState(false);
235
236     // Baseline detail for FTE calculation
237     const [baselineDetail, setBaselineDetail] =
238       useState<BaselineDetail | null>(null);
239
240     // Sorting state for forecast grid
241     const [sortDirection, setSortDirection] = useState<"asc" | "desc">("asc");
242
243     // State for controlling rubros grid collapsible (TODOS view)
244     const [isRubrosGridOpen, setIsRubrosGridOpen] = useState(false);
245
246     // Breakdown view mode for TODOS/portfolio view (Proyectos vs Rubros)
247     // The ForecastRubrosTable component has its own internal viewMode that switches between
248     // 'category' and 'project' views. This state tracks the user's preference at the page level.
249     const [breakdownMode, setBreakdownMode] = useState<'project' | 'rubros'>(() => {
250       const stored = sessionStorage.getItem('forecastBreakdownMode');
251       return stored === 'rubros' ? 'rubros' : 'project';
252     });
253
254     // Handler for breakdown mode changes
255     const handleBreakdownModeChange = (newMode: 'project' | 'rubros') => {
256       setBreakdownMode(newMode);
257       sessionStorage.setItem('forecastBreakdownMode', newMode);
```

```
145     const [savingActuals, setSavingActuals] = useState(false);
146     const [dirtyForecasts, setDirtyForecasts] = useState<Record<string, ForecastRow>>({});
147     const [savingForecasts, setSavingForecasts] = useState(false);
148
149     // Baseline detail and rubros source tracking
150     const [baselineDetail, setBaselineDetail] = useState<BaselineDetailResponse | null>(null);
151     const [rubrosSource, setRubrosSource] = useState<'api' | 'fallback' | null>(null);
152
153     // Request ID tracking to prevent race conditions
154     const requestIdRef = useRef(0);
155
156     // Sorting state for forecast grid
157     const [sortDirection, setSortDirection] = useState<'asc' | 'desc'>('asc');
158
159     // State for controlling rubros grid collapsible (TODOS view)
160     const [isRubrosGridOpen, setIsRubrosGridOpen] = useState(false);
161
```

```
258       // Note: The ForecastRubrosTable component manages its own viewMode internally
259       // and persists it in sessionStorage. This handler updates the page-level state
260       // which could be used to control other aspects of the view in the future.
261     };
262
263     // Stale response guard: Track latest request to prevent race conditions
264     const latestRequestKeyRef = useRef<string>("");
265     const abortControllerRef = useRef<AbortController | null>(null);
266
267     // Ref for scrolling to rubros section when category is clicked
268     const rubrosSectionRef = useRef<HTMLDivElement>(null);
269
270     const [budgetSimulation, setBudgetSimulation] =
271       useState<BudgetSimulationState>({
272         enabled: false,
273         budgetTotal: "",
274         factor: 1.0,
275         estimatedOverride: "",
276       });
277     // Annual Budget state
278     const [budgetYear, setBudgetYear] = useState<number>(
279       new Date().getFullYear()
280     );
281     const [budgetAmount, setBudgetAmount] = useState<string>("");
282     const [budgetCurrency, setBudgetCurrency] = useState<string>("USD");
283     const [budgetLastUpdated, setBudgetLastUpdated] = useState<string | null>(
284       null
285     );
286     const [budgetMissingYear, setBudgetMissingYear] = useState<number | null>(
287       null
288     );
289     const [loadingBudget, setLoadingBudget] = useState(false);
```

```
162     // Stale response guard: Track latest request to prevent race conditions
163     const latestRequestKeyRef = useRef<string>('');
164     const abortControllerRef = useRef<AbortController | null>(null);
165
166     // Ref for scrolling to rubros section when category is clicked
167     const rubrosSectionRef = useRef<HTMLDivElement>(null);
168
169     const [budgetSimulation, setBudgetSimulation] = useState<BudgetSimulationState>({
170       enabled: false,
171       budgetTotal: '',
172       factor: 1.0,
173       estimatedOverride: '',
174     });
175     // Annual Budget state
176     const [budgetYear, setBudgetYear] = useState<number>(new Date().getFullYear());
177     const [budgetAmount, setBudgetAmount] = useState<string>('');
178     const [budgetCurrency, setBudgetCurrency] = useState<string>('USD');
179     const [budgetLastUpdated, setBudgetLastUpdated] = useState<string | null>(null);
180     const [budgetMissingYear, setBudgetMissingYear] = useState<number | null>(null);
181     const [loadingBudget, setLoadingBudget] = useState(false);
```

```
290     const [savingBudget, setSavingBudget] = useState(false);
291     // Monthly Budget state (new - per user request)
292     const [monthlyBudgets, setMonthlyBudgets] =
        useState<MonthlyBudgetInput[]>(
293       []
294     );
295     const [useMonthlyBudget, setUseMonthlyBudget] =
        useState(false);
296     const [loadingMonthlyBudget, setLoadingMonthlyBudget] =
        useState(false);
297     const [savingMonthlyBudget, setSavingMonthlyBudget] =
        useState(false);
298     const [monthlyBudgetLastUpdated,
        setMonthlyBudgetLastUpdated] = useState<
299       string | null
300     >(null);
301     const [monthlyBudgetUpdatedBy,
        setMonthlyBudgetUpdatedBy] = useState<
302       string | null
303     >(null);
304     // Budget Overview state for KPIs
305     const [budgetOverview, setBudgetOverview] = useState<{
306       year: number;
    @@ -316,152 +202,140 @@ export function SDMTForecast() {
316       };
317     } | null>(null);
318     const { user, login } = useAuth();
319     const {
320       selectedProjectId,
321       setSelectedProjectId,
322       selectedPeriod,
323       currentProject,
324       projectChangeCount,
325       projects,
326     } = useProject();
327     const navigate = useNavigate();
328     const location = useLocation();
329     const {
330       lineItems,
331       taxonomyByRubroId,
```

```
182     const [savingBudget, setSavingBudget] = useState(false);
183     // Monthly Budget state (new - per user request)
184     const [monthlyBudgets, setMonthlyBudgets] = useState<MonthlyBudgetInput[]>
        ([]);
185     const [useMonthlyBudget, setUseMonthlyBudget] = useState(false);
186     const [loadingMonthlyBudget, setLoadingMonthlyBudget] = useState(false);
187     const [savingMonthlyBudget, setSavingMonthlyBudget] = useState(false);
188     const [monthlyBudgetLastUpdated, setMonthlyBudgetLastUpdated] =
        useState<string | null>(null);
189     const [monthlyBudgetUpdatedBy, setMonthlyBudgetUpdatedBy] = useState<string |
        null>(null);
190     // Budget Overview state for KPIs
191     const [budgetOverview, setBudgetOverview] = useState<{
192       year: number;
202       };
203     } | null>(null);
204     const { user, login } = useAuth();
205     const { selectedProjectId, setSelectedProjectId, selectedPeriod,
        currentProject, projectChangeCount, projects } = useProject();
206     const navigate = useNavigate();
207     const location = useLocation();
208     const {
209       lineItems,
```

```
332       isLoading: isLineItemsLoading,
333       error: lineItemsError,
334     } = useProjectLineItems({
335       useFallback: true,
336       baselineId: currentProject?.baselineId,
337       withTaxonomy: true,
338     });
339     const safeLineItems = useMemo(
340       () => (Array.isArray(lineItems) ? lineItems : []),
341       [lineItems]
342     );
343     const isPortfolioView = selectedProjectId === ALL_PROJECTS_ID;
344     const forecastMode: "single" | "all-projects" = isPortfolioView
345       ? "all-projects"
346       : "single";
347     const lineItemsForGrid = isPortfolioView ? portfolioLineItems : safeLineItems;
348     const projectStartDate = (currentProject as { start_date?: string } | null)
349       ?.start_date;
350
351     // Helper function to get current month index (1-12) based on today's date and project start
352     const getCurrentMonthIndex = (): number => {
353       const today = new Date();
354       const currentYear = today.getFullYear();
355       const currentMonth = today.getMonth() + 1; // 1-12
356
357       // For portfolio view or when no project start date, use calendar month (1-12)
358       if (!projectStartDate || isPortfolioView) {
359         return currentMonth;
360       }
361
362       // Calculate month index relative to project start date
363       const startDate = new Date(projectStartDate);
364       const startYear = startDate.getFullYear();
365       const startMonth = startDate.getMonth() + 1; // 1-12
```

```
210       isLoading: isLineItemsLoading,
211       error: lineItemsError,
212     } = useProjectLineItems({ useFallback: true, baselineId: currentProject?.baselineId });
213     const safeLineItems = useMemo(
214       () => (Array.isArray(lineItems) ? lineItems : []),
215       [lineItems]
216     );
217     const isPortfolioView = selectedProjectId === ALL_PROJECTS_ID;
218     const forecastMode: 'single' | 'all-projects' = isPortfolioView ? 'all-projects' : 'single';
219     const lineItemsForGrid = isPortfolioView ? portfolioLineItems : safeLineItems;
220     const projectStartDate = (currentProject as { start_date?: string } | null)?.start_date;
221
222     // Helper function to get current month index (1-12) based on today's date and project start
223     const getCurrentMonthIndex = (): number => {
224       const today = new Date();
225       const currentYear = today.getFullYear();
226       const currentMonth = today.getMonth() + 1; // 1-12
227
228       // For portfolio view or when no project start date, use calendar month (1-12)
229       if (!projectStartDate || isPortfolioView) {
230         return currentMonth;
231       }
232
233       // Calculate month index relative to project start date
234       const startDate = new Date(projectStartDate);
235       const startYear = startDate.getFullYear();
236       const startMonth = startDate.getMonth() + 1; // 1-12
```

```
366
367       // Calculate months elapsed since project start
368       const monthsElapsed =
369         (currentYear - startYear) * 12 + (currentMonth -
 startMonth) + 1;
370
371       // Clamp according to baseline duration (fallback to
 60)
372       return clampMonthIndex(monthsElapsed, baselineDetail);
373     };
374
375     // Helper function to compute calendar month from
 monthIndex and project start date
376     const getCalendarMonth = (monthIndex: number): string =>
 {
377       if (!projectStartDate) {
378         // Fallback: display just the month name without
 year for consistency
379         const monthNames = [
380           "Jan",
381           "Feb",
382           "Mar",
383           "Apr",
384           "May",
385           "Jun",
386           "Jul",
387           "Aug",
388           "Sep",
389           "Oct",
390           "Nov",
391           "Dec",
392         ];
393         return monthNames[monthIndex - 1] ||
 `M${monthIndex}`;
394       }
395
396       const startDate = new Date(projectStartDate);
397       startDate.setUTCMonth(startDate.getUTCMonth() +
 (monthIndex - 1));
398       const year = startDate.getUTCFullYear();
399       const month = startDate.getUTCMonth() + 1;
```

```
237
238       // Calculate months elapsed since project start
239       const monthsElapsed = (currentYear - startYear) * 12 + (currentMonth -
 startMonth) + 1;
240
241       // Clamp to 1–12 range (project month index)
242       return Math.max(1, Math.min(12, monthsElapsed));
243     };
244
245     // Helper function to compute calendar month from monthIndex and project
 start date
246     const getCalendarMonth = (monthIndex: number): string => {
247       if (!projectStartDate) {
248         // Fallback: display just the month name without year for consistency
249         const monthNames = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'];
250         return monthNames[monthIndex - 1] || `M${monthIndex}`;
251       }
252
253       const startDate = new Date(projectStartDate);
254       startDate.setUTCMonth(startDate.getUTCMonth() + (monthIndex - 1));
255       const year = startDate.getUTCFullYear();
256       const month = startDate.getUTCMonth() + 1;
```

```
400
401       // Return month name for display (e.g., "May 2025")
402       const monthNames = [
403         "Jan",
404         "Feb",
405         "Mar",
406         "Apr",
407         "May",
408         "Jun",
409         "Jul",
410         "Aug",
411         "Sep",
412         "Oct",
413         "Nov",
414         "Dec",
415       ];
416       return `${monthNames[month - 1]} ${year}`;
417     };
418
```

```
257
258       // Return month name for display (e.g., "May 2025")
259       const monthNames = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug',
'Sep', 'Oct', 'Nov', 'Dec'];
260       return `${monthNames[month - 1]} ${year}`;
261     };
262
263     // Load data when project or period changes
264     useEffect(() => {
265       if (selectedProjectId) {
266         console.log('🔄 Forecast: Loading data for project:', selectedProjectId,
'change count:', projectChangeCount, 'baseline:', currentProject?.baselineId);
267
268         // Abort any previous request
269         if (abortControllerRef.current) {
270           abortControllerRef.current.abort();
271         }
272
273         // Reset state before loading new data
274         setForecastData([]);
275         setPortfolioLineItems([]);
276         loadForecastData();
277       }
278
279       // Cleanup: abort on unmount or when dependencies change
280       return () => {
```

```
                                                    281         if (abortControllerRef.current) {
                                                    282           abortControllerRef.current.abort();
                                                    283         }
                                                    284       };
                                                    285     }, [selectedProjectId, selectedPeriod, projectChangeCount,
                                                        currentProject?.baselineId]);
                                                    286
                                                    287     // Reload data when returning from reconciliation with refresh parameter
                                                    288     useEffect(() => {
                                                    289       const urlParams = new URLSearchParams(location.search);
                                                    290       const refreshParam = urlParams.get('_refresh');
                                                    291       if (refreshParam && selectedProjectId) {
                                                    292         if (import.meta.env.DEV) {
                                                    293           console.log('🔄 Forecast: Refreshing after reconciliation');
                                                    294         }
                                                    295         loadForecastData();
                                                    296       }
                                                    297     }, [location.search]);
                                                    298
419     useEffect(() => {                           299     useEffect(() => {
420       if (!lineItemsError) return;              300       if (!lineItemsError) return;
421                                                 301
422       const message = handleFinanzasApiError(lineItemsError,    302       const message = handleFinanzasApiError(lineItemsError, {
        {
423         onAuthError: login,                     303         onAuthError: login,
424         fallback: "No se pudieron cargar los rubros para    304         fallback: 'No se pudieron cargar los rubros para forecast.',
        forecast.",
425       });                                       305       });
426       setForecastError((prev) => prev || message);    306       setForecastError((prev) => prev || message);
427     }, [lineItemsError, login]);                307     }, [lineItemsError, login]);
428                                                 308
429     // Load baseline details for FTE calculation    309     // Load baseline details for FTE calculation
430     useEffect(() => {                           310     useEffect(() => {
431       if (currentProject?.baselineId && !isPortfolioView) {    311       if (currentProject?.baselineId && !isPortfolioView) {
432         getBaselineById(currentProject.baselineId)    312         getBaselineById(currentProject.baselineId, currentProject.id)
433           .then((data) => {                         313           .then((data) => {
434             if (import.meta.env.DEV) {              314             if (import.meta.env.DEV) {
435               console.log(                          315               console.log('[SDMTForecast] Baseline details loaded for FTE
                                                        calculation');
436                 "[SDMTForecast] Baseline details loaded for
        FTE calculation"
```

```
437                );
438              }
439              setBaselineDetail(data);
440            })
441            .catch((err) => {
442              console.error("Failed to load baseline
    details:", err);
443              setBaselineDetail(null);
444            });
445        } else {
446          setBaselineDetail(null);
447        }
448      }, [currentProject?.baselineId, isPortfolioView]);
449
450      const loadForecastData = useCallback(async () => {
451        // If we're in portfolio (TODOS) view, we MUST load
    portfolio-wide forecast even when there is
452        // no selectedProjectId. Only skip forecast load when
    not in portfolio view and no project selected.
453        if (!isPortfolioView && !selectedProjectId) {
454          console.log("❌ No project selected and not
    portfolio view, skipping forecast load");
455          return;
456        }
457
458        // Create a new abort controller for this request
459        abortControllerRef.current = new AbortController();
460
461        // Generate unique request key to identify this
    specific request
462        const requestKey = `${selectedProjectId}__${
463          currentProject?.baselineId || ""
464        }__${Date.now()}`;
465        latestRequestKeyRef.current = requestKey;
466
467        try {
    @@ -470,14 +344,14 @@ export function SDMTForecast() {
470          setForecastError(null);
471          setDirtyActuals({});
472          setDirtyForecasts({});
```

```
316              }
317              setBaselineDetail(data);
318            })
319            .catch((err) => {
320              console.error('Failed to load baseline details:', err);
321              setBaselineDetail(null);
322            });
323        } else {
324          setBaselineDetail(null);
325        }
326      }, [currentProject?.baselineId, isPortfolioView]);
327
328      const loadForecastData = async () => {
329        if (!selectedProjectId) {
330          console.log('❌ No project selected, skipping forecast load');
331          return;
332        }
333
334        // Create a new abort controller for this request
335        abortControllerRef.current = new AbortController();
336
337        // Generate unique request key to identify this specific request
338        const requestKey = `${selectedProjectId}__${currentProject?.baselineId ||
    ''}__${Date.now()}`;
339        latestRequestKeyRef.current = requestKey;
340
341        try {
344          setForecastError(null);
345          setDirtyActuals({});
346          setDirtyForecasts({});
```

```
473
474         // Handle CURRENT_MONTH period - always load 12 months but filter to current month later
475         const isCurrentMonthMode = selectedPeriod === "CURRENT_MONTH";
476         const months = isCurrentMonthMode ? 12 : parseInt(selectedPeriod);
477
478         if (import.meta.env.DEV) {
479           console.debug("[Forecast] Loading data", {
480             projectId: selectedProjectId,
481             months,
482             isCurrentMonthMode,
483             selectedPeriod,
    @@ -488,38 +362,39 @@ export function SDMTForecast() {
488         if (isPortfolioView) {
489           await loadPortfolioForecast(months, requestKey);
490         } else {
491           await loadSingleProjectForecast(selectedProjectId, months, requestKey);
492         }
493
494         // Verify this is still the latest request before applying results
495         if (latestRequestKeyRef.current !== requestKey) {
496           if (import.meta.env.DEV) {
497             console.debug("[Forecast] Discarding stale response", {
498               requestKey,
499               latest: latestRequestKeyRef.current,
500             });
501           }
502           return; // Stale response, ignore it
```

```
347
348         // Handle CURRENT_MONTH period - always load 12 months but filter to current month later
349         const isCurrentMonthMode = selectedPeriod === 'CURRENT_MONTH';
350         const months = isCurrentMonthMode ? 12 : parseInt(selectedPeriod);
351
352         if (import.meta.env.DEV) {
353           console.debug('[Forecast] Loading data', {
354             projectId: selectedProjectId,
355             months,
356             isCurrentMonthMode,
357             selectedPeriod,
362         if (isPortfolioView) {
363           await loadPortfolioForecast(months, requestKey);
364         } else {
365           // Load baseline rubros for single project (non-portfolio) view
366           if (currentProject?.baselineId) {
367             await loadBaselineRubros(
368               selectedProjectId,
369               currentProject.baselineId,
370               abortControllerRef.current?.signal
371             );
372           }
373           await loadSingleProjectForecast(selectedProjectId, months, requestKey);
374         }
375
376         // Verify this is still the latest request before applying results
377         if (latestRequestKeyRef.current !== requestKey) {
378           if (import.meta.env.DEV) {
379             console.debug('[Forecast] Discarding stale response', { requestKey, latest: latestRequestKeyRef.current });
380           }
381           return; // Stale response, ignore it
```

```
503         }
504       } catch (error) {
505         // Ignore aborted requests
506         if (error instanceof Error && error.name === "AbortError") {
507           if (import.meta.env.DEV) {
508             console.debug("[Forecast] Request aborted", { requestKey });
509           }
510           return;
511         }
512
513         console.error(
514           "❌ Failed to load forecast data for project:",
515           selectedProjectId,
516           error
517         );
518         const message = handleFinanzasApiError(error, {
519           onAuthError: login,
520           fallback: "No se pudo cargar el forecast.",
521         });
522
523         // Only set error if this is still the latest request
524         if (latestRequestKeyRef.current === requestKey) {
525           setForecastError(message);
@@ -532,7 +407,7 @@ export function SDMTForecast() {
532           setIsLoadingForecast(false);
533         }
534       }
535     }, [isPortfolioView, selectedProjectId, currentProject?.baselineId, selectedPeriod, login]);
536
537     const transformLineItemsToForecast = (
538       lineItems: LineItemLike[],
@@ -549,9 +424,7 @@ export function SDMTForecast() {
549         return Number.isFinite(numeric) ? numeric : fallback;
550       };
551
```

```
382         }
383       } catch (error) {
384         // Ignore aborted requests
385         if (error instanceof Error && error.name === 'AbortError') {
386           if (import.meta.env.DEV) {
387             console.debug('[Forecast] Request aborted', { requestKey });
388           }
389           return;
390         }
391
392         console.error('❌ Failed to load forecast data for project:', selectedProjectId, error);
393         const message = handleFinanzasApiError(error, {
394           onAuthError: login,
395           fallback: 'No se pudo cargar el forecast.',
396         });
397
398         // Only set error if this is still the latest request
399         if (latestRequestKeyRef.current === requestKey) {
400           setForecastError(message);
@@ -532,7 +407,7 @@ export function SDMTForecast() {
407           setIsLoadingForecast(false);
408         }
409       }
410     };
411
412     const transformLineItemsToForecast = (
413       lineItems: LineItemLike[],
@@ -549,9 +424,7 @@ export function SDMTForecast() {
424         return Number.isFinite(numeric) ? numeric : fallback;
425       };
426
```

```
552      const resolveDuration = (
553        item: LineItemLike
554      ): { value: number; fromData: boolean } => {
555        const baseline = item.baseline as LineItemLike | undefined;
556        const candidates = [
557          item.duration,
     @@ -571,20 +444,20 @@ export function SDMTForecast() {
571      };
572
573      const resolveString = (value: unknown): string | undefined => {
574        if (typeof value === "string" && value.trim().length > 0) {
575          return value;
576        }
577        return undefined;
578      };
579
580      lineItems.forEach((item) => {
581        const lineItemId =
582          resolveString(item.id) ||
583          resolveString(item.rubroId) ||
584          resolveString(item.rubro_id) ||
585          resolveString(item.linea_codigo) ||
586          resolveString(item.linea_id) ||
587          "";
588
589        if (!lineItemId) {
590          return;
     @@ -599,33 +472,20 @@ export function SDMTForecast() {
599        const duration = resolveDuration(item);
600        const startMonthRaw = resolveNumber(item.start_month, NaN);
601        const endMonthRaw = resolveNumber(item.end_month, NaN);
602        const hasExplicitRange =
603          Number.isFinite(startMonthRaw) || Number.isFinite(endMonthRaw);
604        const startMonth = hasExplicitRange
```

```
427      const resolveDuration = (item: LineItemLike): { value: number; fromData: boolean } => {
428        const baseline = item.baseline as LineItemLike | undefined;
429        const candidates = [
430          item.duration,
444      };
445
446      const resolveString = (value: unknown): string | undefined => {
447        if (typeof value === 'string' && value.trim().length > 0) {
448          return value;
449        }
450        return undefined;
451      };
452
453      lineItems.forEach(item => {
454        const lineItemId =
455          resolveString(item.id) ||
456          resolveString(item.rubroId) ||
457          resolveString(item.rubro_id) ||
458          resolveString(item.linea_codigo) ||
459          resolveString(item.linea_id) ||
460          '';
461
462        if (!lineItemId) {
463          return;
472        const duration = resolveDuration(item);
473        const startMonthRaw = resolveNumber(item.start_month, NaN);
474        const endMonthRaw = resolveNumber(item.end_month, NaN);
475        const hasExplicitRange = Number.isFinite(startMonthRaw) || Number.isFinite(endMonthRaw);
476        const startMonth = hasExplicitRange
```

```
605          ? Math.max(
606              1,
607              Math.min(
608                months,
609                Number.isFinite(startMonthRaw) ? Math.trunc(startMonthRaw) : 1
610              )
611            )
612          : 1;
613        const endMonth = hasExplicitRange
614          ? Math.max(
615              startMonth,
616              Math.min(
617                months,
618                Number.isFinite(endMonthRaw) ? Math.trunc(endMonthRaw) : months
619              )
620            )
621          : duration.fromData
622          ? Math.min(months, duration.value)
623          : months;
624
625        const resolveFirstNumber = (values: unknown[]): number | null => {
626          for (const value of values) {
627            if (value === null || value === undefined) continue;
628            if (typeof value === "string" && value.trim().length === 0) continue;
629            const numeric = Number(value);
630            if (Number.isFinite(numeric)) return numeric;
631          }
```

```
@@ -645,8 +505,7 @@ export function SDMTForecast() {
```

```
645          item.unitCost ?? item.costo_unitario ?? item.unit_cost,
646          0
647        );
648        const qty =
649          resolveNumber(item.qty ?? item.cantidad ?? item.quantity, 1) || 1;
```

```
477          ? Math.max(1, Math.min(months, Number.isFinite(startMonthRaw) ? Math.trunc(startMonthRaw) : 1))
478          : 1;
479        const endMonth = hasExplicitRange
480          ? Math.max(startMonth, Math.min(months, Number.isFinite(endMonthRaw) ? Math.trunc(endMonthRaw) : months))
481          : duration.fromData
482          ? Math.min(months, duration.value)
483          : months;
484
485        const resolveFirstNumber = (values: unknown[]): number | null => {
486          for (const value of values) {
487            if (value === null || value === undefined) continue;
488            if (typeof value === 'string' && value.trim().length === 0) continue;
489            const numeric = Number(value);
490            if (Number.isFinite(numeric)) return numeric;
491          }
```

```
505          item.unitCost ?? item.costo_unitario ?? item.unit_cost,
506          0
507        );
508        const qty = resolveNumber(item.qty ?? item.cantidad ?? item.quantity, 1) || 1;
```

```
650         const computedTotal = baseTotal ?? unitCost * qty;
651         const activeMonthCount = Math.max(1, endMonth - startMonth + 1);
652         const defaultMonthlyAmount = computedTotal / activeMonthCount;
...   @@ -661,33 +520,19 @@ export function SDMTForecast() {
661             ? monthlyOverride
662             : defaultMonthlyAmount;
663
664         // Try to resolve category and description from canonical taxonomy
665         const taxonomy = getTaxonomyById(lineItemId) || getTaxonomyById(rubroKey);
666
667         const description =
668           resolveString(item.description) ||
669           resolveString(item.descripcion) ||
670           (taxonomy ? taxonomy.linea_gasto || taxonomy.descripcion : '') ||
671           lineItemId;
672
673         const category =
674           resolveString(item.category) ||
675           resolveString(item.categoria) ||
676           (taxonomy ? taxonomy.categoria : '');
677
678         forecastCells.push({
679           line_item_id: lineItemId,
680           rubroId: rubroKey,
681           projectId,
682           description,
683           category,
684           month,
685           planned: amount,
686           forecast: amount,
687           actual: 0,
688           variance: 0,
689           last_updated: new Date().toISOString(),
690           updated_by: user?.email || user?.login || "system",
```

```
509         const computedTotal = baseTotal ?? unitCost * qty;
510         const activeMonthCount = Math.max(1, endMonth - startMonth + 1);
511         const defaultMonthlyAmount = computedTotal / activeMonthCount;
520             ? monthlyOverride
521             : defaultMonthlyAmount;
522
523           forecastCells.push({
524             line_item_id: lineItemId,
525             rubroId: rubroKey,
526             projectId,
527             description: resolveString(item.description) || resolveString(item.descripcion) || '',
528             category: resolveString(item.category) || resolveString(item.categoria) || '',
529             month,
530             planned: amount,
531             forecast: amount,
532             actual: 0,
533             variance: 0,
534             last_updated: new Date().toISOString(),
535             updated_by: user?.email || user?.login || 'system',
```

```
691           });
692         }
693       });
    @@ -702,35 +547,125 @@ export function SDMTForecast() {
702     };
703
704     /**
705      * Load forecast data for a single project
706      */
707     const loadSingleProjectForecast = async (
708       projectId: string,
709       months: number,
710       requestKey: string
711     ) => {
712       const payload = await getForecastPayload(projectId, months);
```

```
536           });
537         }
538       });
547     };
548
549     /**
550      * Load baseline rubros with fallback to allocations/prefacturas
551      * Uses Promise.allSettled to fetch all data in parallel and handle partial failures
552      */
553     const loadBaselineRubros = async (projectId: string, baselineId: string, signal?: AbortSignal) => {
554       if (!projectId || !baselineId) {
555         if (import.meta.env.DEV) {
556           console.debug('[loadBaselineRubros] Skipping: missing projectId or baselineId');
557         }
558         return;
559       }
560
561       const myRequestId = ++requestIdRef.current;
562
563       if (import.meta.env.DEV) {
564         console.debug(`[loadBaselineRubros] Starting request ${myRequestId} for baseline ${baselineId}`);
565       }
566
567       // Reset rubros source while loading
568       setRubrosSource(null);
569
570       try {
571         // Fetch all data in parallel using Promise.allSettled
572         const tasks = [
573           finanzasClient.getRubrosForBaseline(projectId, baselineId, { signal }),
574           finanzasClient.getAllocationsForBaseline(projectId, baselineId, { signal }),
575           finanzasClient.getPrefacturasForBaseline(projectId, baselineId, { signal }),
576           finanzasClient.getBaselineById(baselineId, { signal, projectId }),
577         ];
```

```
578
579       const results = await Promise.allSettled(tasks);
580
581       // Ensure this request is still active
582       if (myRequestId !== requestIdRef.current) {
583         if (import.meta.env.DEV) {
584           console.debug(`[loadBaselineRubros] Request ${myRequestId} stale, aborting`);
585         }
586         return;
587       }
588
589       // Extract results safely with strong type guards
590       const rubrosRes = results[0].status === 'fulfilled' && Array.isArray(results[0].value) ? results[0].value as any[] : [];
591       const allocationsRes = results[1].status === 'fulfilled' && Array.isArray(results[1].value) ? results[1].value as any[] : [];
592       const prefacturasRes = results[2].status === 'fulfilled' && Array.isArray(results[2].value) ? results[2].value as any[] : [];
593       const baselineRes: BaselineDetailResponse | null = (results[3] && results[3].status === 'fulfilled') ? results[3].value as BaselineDetailResponse : null;
594
595       // Update baseline detail
596       setBaselineDetail(baselineRes);
597
598       if (import.meta.env.DEV) {
599         console.debug(`[loadBaselineRubros] Results:`, {
600           rubros: rubrosRes?.length || 0,
601           allocations: allocationsRes?.length || 0,
602           prefacturas: prefacturasRes?.length || 0,
603           baseline: baselineRes ? 'loaded' : 'null',
604         });
605       }
606
607       // If rubros endpoint returned data, use it
608       if (rubrosRes && rubrosRes.length > 0) {
609         setRubrosSource('api');
610         if (import.meta.env.DEV) {
611           console.debug(`[loadBaselineRubros] Using ${rubrosRes.length} rubros from API`);
```

```
612         }
613         return rubrosRes;
614       }
615
616       // Fallback: materialize from allocations & prefacturas if any present
617       if ((allocationsRes && allocationsRes.length > 0) || (prefacturasRes &&
    prefacturasRes.length > 0)) {
618         const materialized = rubrosFromAllocations(allocationsRes || [],
    prefacturasRes || []);
619         setRubrosSource('fallback');
620         if (import.meta.env.DEV) {
621           console.debug(
622             `[loadBaselineRubros] Materialized ${materialized.length} rubros
    from ${allocationsRes?.length || 0} allocations + ${prefacturasRes?.length ||
    0} prefacturas`
623           );
624         }
625         return materialized;
626       }
627
628       // Nothing available
629       setRubrosSource(null);
630       if (import.meta.env.DEV) {
631         console.debug('[loadBaselineRubros] No rubros, allocations, or
    prefacturas available');
632       }
633       return [];
634     } catch (error) {
635       // Handle AbortError gracefully
636       if (error instanceof Error && error.name === 'AbortError') {
637         if (import.meta.env.DEV) {
638           console.debug(`[loadBaselineRubros] Request ${myRequestId} aborted`);
639         }
640         return [];
641       }
642
643       console.error('[loadBaselineRubros] Error loading baseline rubros:',
    error);
644       setRubrosSource(null);
645       return [];
646     }
```

713

```
                                                                  647       };
                                                                  648
                                                                  649       const loadSingleProjectForecast = async (projectId: string, months: number, requestKey: string) => {
                                                                  650         const payload = await getForecastPayload(projectId, months);
                                                                  651
714         // Check if request is still valid before continuing  652         // Check if request is still valid before continuing
715         if (latestRequestKeyRef.current !== requestKey) {    653         if (latestRequestKeyRef.current !== requestKey) {
716           return; // Stale, abort processing                 654           return; // Stale, abort processing
717         }                                                    655         }
718                                                              656
719         // Pass baseline ID and enable debug mode in development for better diagnostics
                                                                  657         // Pass baseline ID and enable debug mode in development for better diagnostics
720         const debugMode = import.meta.env.DEV;               658         const debugMode = import.meta.env.DEV;
721         let normalized = normalizeForecastCells(payload.data, {
                                                                  659         let normalized = normalizeForecastCells(payload.data, {
722           baselineId: currentProject?.baselineId,            660           baselineId: currentProject?.baselineId,
723           debugMode,                                         661           debugMode
724         });                                                  662         });
725         let usedFallback = false;                            663         let usedFallback = false;
726         const baselineStatus = resolveBaselineStatus(        664         const baselineStatus = resolveBaselineStatus(
727           currentProject as {                                665           currentProject as { baselineStatus?: string; baseline_status?: string } | null
728             baselineStatus?: string;
729             baseline_status?: string;
730           } | null
731         );                                                   666         );
732         const hasAcceptedBaseline = baselineStatus === "accepted";
                                                                  667         const hasAcceptedBaseline = baselineStatus === 'accepted';
733                                                              668
734         // Fallback: If server forecast is empty and we have line items, use them
                                                                  669         // Fallback: If server forecast is empty and we have line items, use them
735         if (                                                 670         if (
736           (!normalized || normalized.length === 0) &&        671           (!normalized || normalized.length === 0) &&
•••   @@ -743,43 +678,38 @@ export function SDMTForecast() {
743             `[SDMTForecast] Using baseline fallback for ${projectId}, baseline ${currentProject?.baselineId}: ${safeLineItems.length} line items`
                                                                  678             `[SDMTForecast] Using baseline fallback for ${projectId}, baseline ${currentProject?.baselineId}: ${safeLineItems.length} line items`
744           );                                                 679           );
745         }                                                    680         }
746         normalized = transformLineItemsToForecast(           681         normalized = transformLineItemsToForecast(safeLineItems, months, projectId);
```

```
747           safeLineItems,
748           months,
749           projectId
750         );
751         usedFallback = true;
752       } else if (normalized && normalized.length > 0 && import.meta.env.DEV) {
753         console.debug(
754           `[SDMTForecast] Using server forecast rows for ${projectId}, baseline ${currentProject?.baselineId}`
755         );
756       }
757
758       setDataSource(usedFallback ? "mock" : payload.source); // Mark as 'mock' to indicate fallback
759       setGeneratedAt(payload.generatedAt);
760       setPortfolioLineItems([]);
761
762       // Get matched invoices and sync with actuals
763       const invoices = await getProjectInvoices(projectId);
764
765       // Check again after async operation
766       if (latestRequestKeyRef.current !== requestKey) {
767         return;
768       }
769
770       const matchedInvoices = invoices.filter((inv) => inv.status === "Matched");
771
772       const updatedData: ForecastRow[] = normalized.map((cell) => {
773         const matchedInvoice = matchedInvoices.find(
774           (inv) =>
775             inv.line_item_id === cell.line_item_id && inv.month === cell.month
776         );
777
778         const withActuals = matchedInvoice
779           ? {
```

```
682         usedFallback = true;
683       } else if (normalized && normalized.length > 0 && import.meta.env.DEV) {
684         console.debug(
685           `[SDMTForecast] Using server forecast rows for ${projectId}, baseline ${currentProject?.baselineId}`
686         );
687       }
688
689       setDataSource(usedFallback ? 'mock' : payload.source); // Mark as 'mock' to indicate fallback
690       setGeneratedAt(payload.generatedAt);
691       setPortfolioLineItems([]);
692
693       // Get matched invoices and sync with actuals
694       const invoices = await getProjectInvoices(projectId);
695
696       // Check again after async operation
697       if (latestRequestKeyRef.current !== requestKey) {
698         return;
699       }
700
701       const matchedInvoices = invoices.filter(inv => inv.status === 'Matched');
702
703       const updatedData: ForecastRow[] = normalized.map(cell => {
704         const matchedInvoice = matchedInvoices.find(inv =>
705           inv.line_item_id === cell.line_item_id && inv.month === cell.month
706         );
707
708         const withActuals = matchedInvoice
709           ? {
```

```
780                ...cell,
781                actual: matchedInvoice.amount || 0,
782                variance: cell.forecast - cell.planned, // Keep forecast-based variance
783              }
784            : cell;
785
...  @@ -791,7 +721,7 @@ export function SDMTForecast() {
791      });
792
793      if (import.meta.env.DEV) {
794        console.debug("[Forecast] data pipeline", {
795          projectId,
796          rawCells: Array.isArray(payload.data) ? payload.data.length : 0,
797          normalizedCells: normalized.length,
...  @@ -811,24 +741,16 @@ export function SDMTForecast() {
811      setForecastData(updatedData);
812
813      if (import.meta.env.DEV) {
814        console.debug("[Forecast] Data loaded", {
815          projectId,
816          source: usedFallback ? "lineItems-fallback" : payload.source,
817          records: updatedData.length,
818        });
819      }
820    };
821
822    const loadPortfolioForecast = async (months: number, requestKey: string) => {
823      if (import.meta.env.DEV) {
824        console.log(
825          `[loadPortfolioForecast] projects loaded: ${projects.length}, ids: ${JSON.stringify(projects.map(p => p.id))}`
826        );
827      }
828
829      const candidateProjects = projects.filter(
```

```
710                ...cell,
711                actual: matchedInvoice.amount || 0,
712                variance: cell.forecast - cell.planned // Keep forecast-based variance
713              }
714            : cell;
715
721      });
722
723      if (import.meta.env.DEV) {
724        console.debug('[Forecast] data pipeline', {
725          projectId,
726          rawCells: Array.isArray(payload.data) ? payload.data.length : 0,
727          normalizedCells: normalized.length,
741      setForecastData(updatedData);
742
743      if (import.meta.env.DEV) {
744        console.debug('[Forecast] Data loaded', {
745          projectId,
746          source: usedFallback ? 'lineItems-fallback' : payload.source,
747          records: updatedData.length,
748        });
749      }
750    };
751
752    const loadPortfolioForecast = async (months: number, requestKey: string) => {
753      const candidateProjects = projects.filter(project => project.id && project.id !== ALL_PROJECTS_ID);
```

```
830       (project) => project.id && project.id !==
 ALL_PROJECTS_ID
831     );
832     // TODO(SDMT): Replace per-project fan-out with
 aggregate portfolio endpoints when available.
833
834     // Guard: Don't error out if projects haven't loaded
 yet
•••   @@ -838,28 +760,26 @@ export function SDMTForecast() {
838       if (projects.length <
 MINIMUM_PROJECTS_FOR_PORTFOLIO) {
839         // Projects might still be loading; don't set
 error yet
840         if (import.meta.env.DEV) {
841           console.debug(
842             "[Forecast] Portfolio: Waiting for projects to
 load..."
843           );
844         }
845         setForecastData([]);
846         return;
847       }
848       // If we have projects but they're all filtered out,
 that's a real empty state
849       setForecastError("No hay proyectos disponibles para
 consolidar.");
850       setForecastData([]);
851       return;
852     }
853
854     const portfolioResults = await Promise.all(
855       candidateProjects.map(async (project) => {
856         try {
857           const [payload, invoices, projectLineItems] =
 await Promise.all([
858             getForecastPayload(project.id, months),
859             getProjectInvoices(project.id),
860             getProjectRubros(project.id).catch(() => [] as
 LineItem[]),
861           ]);
862
863           // Check if request is still valid after async
 operations
```

```
754     // TODO(SDMT): Replace per-project fan-out with aggregate portfolio
 endpoints when available.
755
756     // Guard: Don't error out if projects haven't loaded yet
760       if (projects.length < MINIMUM_PROJECTS_FOR_PORTFOLIO) {
761         // Projects might still be loading; don't set error yet
762         if (import.meta.env.DEV) {
763           console.debug('[Forecast] Portfolio: Waiting for projects to
 load...');
764         }
765         setForecastData([]);
766         return;
767       }
768       // If we have projects but they're all filtered out, that's a real empty
 state
769       setForecastError('No hay proyectos disponibles para consolidar.');
770       setForecastData([]);
771       return;
772     }
773
774     const portfolioResults = await Promise.all(
775       candidateProjects.map(async project => {
776         try {
777           const [payload, invoices, projectLineItems] = await Promise.all([
778             getForecastPayload(project.id, months),
779             getProjectInvoices(project.id),
780             getProjectRubros(project.id).catch(() => [] as LineItem[]),
781           ]);
782
783           // Check if request is still valid after async operations
```

```
864             if (latestRequestKeyRef.current !== requestKey) {
865               // Return empty result to be filtered out, don't throw
    @@ -869,57 +789,36 @@ export function SDMTForecast() {
869             const debugMode = import.meta.env.DEV;
870             let normalized = normalizeForecastCells(payload.data, {
871               baselineId: project.baselineId,
872               debugMode,
873             });
874             const baselineStatus = resolveBaselineStatus(
875               project as {
876                 baselineStatus?: string;
877                 baseline_status?: string;
878               } | null
879             );
880             const hasAcceptedBaseline = baselineStatus === "accepted";
881             let usedFallback = false;
882
883             if (
884               (!normalized || normalized.length === 0) &&
885               projectLineItems.length > 0 &&
886               hasAcceptedBaseline
887             ) {
888               if (import.meta.env.DEV) {
889                 console.debug(
890                   `[SDMTForecast] Using baseline fallback for ${project.id}, baseline ${project.baselineId}: ${projectLineItems.length} line items`
891                 );
892               }
893               normalized = transformLineItemsToForecast(
894                 projectLineItems,
895                 months,
896                 project.id
897               );
898               usedFallback = true;
```

```
784             if (latestRequestKeyRef.current !== requestKey) {
785               // Return empty result to be filtered out, don't throw
789             const debugMode = import.meta.env.DEV;
790             let normalized = normalizeForecastCells(payload.data, {
791               baselineId: project.baselineId,
792               debugMode
793             });
794             const baselineStatus = resolveBaselineStatus(
795               project as { baselineStatus?: string; baseline_status?: string } | null
796             );
797             const hasAcceptedBaseline = baselineStatus === 'accepted';
798             let usedFallback = false;
799
800             if ((!normalized || normalized.length === 0) && projectLineItems.length > 0 && hasAcceptedBaseline) {
801               if (import.meta.env.DEV) {
802                 console.debug(
803                   `[SDMTForecast] Using baseline fallback for ${project.id}, baseline ${project.baselineId}: ${projectLineItems.length} line items`
804                 );
805               }
806               normalized = transformLineItemsToForecast(projectLineItems, months, project.id);
807               usedFallback = true;
```

```
899            } else if (normalized.length > 0 && import.meta.env.DEV) {
900              console.debug(
901                `[SDMTForecast] Using server forecast rows for ${project.id}, baseline ${project.baselineId}`
902              );
903            }
904            const matchedInvoices = invoices.filter(
905              (inv) => inv.status === "Matched"
906            );
907
908            const projectData: ForecastRow[] = normalized.map((cell) => {
909              const matchedInvoice = matchedInvoices.find((inv) => {
910                const invoiceMonth = normalizeInvoiceMonth(inv.month);
911                return (
912                  inv.line_item_id === cell.line_item_id &&
913                  invoiceMonth === cell.month
914                );
915              });
916
917              const withActuals = matchedInvoice
918                ? {
919                    ...cell,
920                    actual: matchedInvoice.amount || 0,
921                    variance: cell.forecast - cell.planned,
922                  }
923                : cell;
924
925              return {
    @@ -929,282 +828,150 @@ export function SDMTForecast() {
929              };
930            });
931
932            if (import.meta.env.DEV) {
933              console.log(
934                `[loadPortfolioForecast] project ${project.id} forecastRows=${projectData.length}
```

```
808            } else if (normalized.length > 0 && import.meta.env.DEV) {
809              console.debug(
810                `[SDMTForecast] Using server forecast rows for ${project.id}, baseline ${project.baselineId}`
811              );
812            }
813            const matchedInvoices = invoices.filter(inv => inv.status === 'Matched');
814
815            const projectData: ForecastRow[] = normalized.map(cell => {
816              const matchedInvoice = matchedInvoices.find(inv =>
817                inv.line_item_id === cell.line_item_id && inv.month === cell.month
818              );
819
820              const withActuals = matchedInvoice
821                ? { ...cell, actual: matchedInvoice.amount || 0, variance: cell.forecast - cell.planned }
822                : cell;
823
824              return {
828              };
829            });
830
```

```
                invoices=${invoices.length}`
935               );
936             }
937
938             return {
939               project,
940               data: projectData,
941               lineItems: projectLineItems.map((item) => ({
942                 ...item,
943                 projectId: project.id,
944                 projectName: project.name,
945               })),
946               generatedAt: payload.generatedAt,
947               usedFallback,
948             };
949           } catch (error) {
950             // Log error but don't crash entire portfolio load
951             console.warn(
952               `[Forecast] Failed to load data for project ${project.id} (${project.name}):`,
953               error
954             );
955             // Return null to be filtered out later
956             return null;
957           }
958         })
959       );
960
961       // Filter out null results from aborted requests
962       const validResults = portfolioResults.filter((result) => result !== null);
963
964       // Final check before setting state
965       if (latestRequestKeyRef.current !== requestKey) {
966         return;
967       }
968
```

```
831             return {
832               project,
833               data: projectData,
834               lineItems: projectLineItems.map(item => ({ ...item, projectId: project.id, projectName: project.name })),
835               generatedAt: payload.generatedAt,
836               usedFallback,
837             };
838           } catch (error) {
839             // Log error but don't crash entire portfolio load
840             console.warn(`[Forecast] Failed to load data for project ${project.id} (${project.name}):`, error);
841             // Return null to be filtered out later
842             return null;
843           }
844         })
845       );
846
847       // Filter out null results from aborted requests
848       const validResults = portfolioResults.filter(result => result !== null);
849
850       // Final check before setting state
851       if (latestRequestKeyRef.current !== requestKey) {
852         return;
853       }
854
```

```
969       const aggregatedData = validResults.flatMap((result) => result.data);
970       const aggregatedLineItems = validResults.flatMap(
971         (result) => result.lineItems
972       );
973       const firstGeneratedAt = validResults.find(
974         (result) => result.generatedAt
975       )?.generatedAt;
976
977       setDataSource("api");
978       setGeneratedAt(firstGeneratedAt || new Date().toISOString());
979       setPortfolioLineItems(aggregatedLineItems);
980       setForecastData(aggregatedData);
981
982       if (import.meta.env.DEV) {
983         console.debug("[Forecast] Portfolio data loaded", {
984           projects: candidateProjects.length,
985           records: aggregatedData.length,
986           lineItems: aggregatedLineItems.length,
987         });
988       }
989     };
990
991     // Consolidated data loading effect: handles initial load, route changes, and event-driven refreshes
992     // This ensures forecast loads on:
993     // 1. Initial mount / when dependencies change (project, period, baseline)
994     // 2. Route change (location.key)
995     // 3. Document visibility change (hidden → visible)
996     // 4. Window focus
997     // 5. Manual refresh via URL parameter (_refresh)
998     useEffect(() => {
999       // Helper to trigger load and guard against overlapping calls
1000      const triggerLoad = () => {
1001        if (selectedProjectId) {
1002          if (import.meta.env.DEV) {
```

```
855       const aggregatedData = validResults.flatMap(result => result.data);
856       const aggregatedLineItems = validResults.flatMap(result => result.lineItems);
857       const firstGeneratedAt = validResults.find(result => result.generatedAt)?.generatedAt;
858
859       setDataSource('api');
860       setGeneratedAt(firstGeneratedAt || new Date().toISOString());
861       setPortfolioLineItems(aggregatedLineItems);
862       setForecastData(aggregatedData);
863
864       if (import.meta.env.DEV) {
865         console.debug('[Forecast] Portfolio data loaded', {
866           projects: candidateProjects.length,
867           records: aggregatedData.length,
868           lineItems: aggregatedLineItems.length,
869         });
870       }
871     };
872
873     const handleCellEdit = (line_item_id: string, month: number, type: 'forecast' | 'actual') => {
874       const cell = forecastData.find(c => c.line_item_id === line_item_id && c.month === month);
```

```
1003            console.log(
1004              "🔄 Forecast: Loading data for project:",
1005              selectedProjectId,
1006              "change count:",
1007              projectChangeCount,
1008              "baseline:",
1009              currentProject?.baselineId
1010            );
1011          }
1012
1013          // Abort any previous request
1014          if (abortControllerRef.current) {
1015            abortControllerRef.current.abort();
1016          }
1017
1018          // Reset state before loading new data
1019          setForecastData([]);
1020          setPortfolioLineItems([]);
1021          loadForecastData();
1022        }
1023      };
1024
1025      // Run once on mount / whenever route (location.key)
     or main deps change
1026      triggerLoad();
1027
1028      // Check for URL refresh parameter
1029      const urlParams = new
     URLSearchParams(location.search);
1030      const refreshParam = urlParams.get("_refresh");
1031      if (refreshParam && selectedProjectId) {
1032        if (import.meta.env.DEV) {
1033          console.log("🔄 Forecast: Refreshing after
     reconciliation (URL param)");
1034        }
1035        // Already triggered above, but this logs the reason
1036      }
1037
1038      // Guard to prevent repeated visibility-based
     refreshes
```

```
1039      let didRefreshOnVisibility = false;
1040
1041      // Visibility change: when tab becomes visible again, reload once
1042      const onVisibility = () => {
1043        if (document.visibilityState === 'visible' && selectedProjectId && !didRefreshOnVisibility) {
1044          didRefreshOnVisibility = true;
1045          if (import.meta.env.DEV) {
1046            console.log("🔄 Forecast: Refreshing on visibility change");
1047          }
1048          triggerLoad();
1049        } else if (document.visibilityState === 'hidden') {
1050          // Reset the flag when tab becomes hidden so next visibility will trigger refresh
1051          didRefreshOnVisibility = false;
1052        }
1053      };
1054      window.addEventListener('visibilitychange', onVisibility);
1055
1056      // Cleanup: remove event listeners and abort outstanding request when component unmounts
1057      return () => {
1058        window.removeEventListener('visibilitychange', onVisibility);
1059        // Abort any outstanding request when component unmounts
1060        if (abortControllerRef.current) {
1061          abortControllerRef.current.abort();
1062        }
1063      };
1064    }, [
1065      loadForecastData,
1066      location.key,
1067      location.search,
1068      selectedProjectId,
1069      selectedPeriod,
1070      projectChangeCount,
1071      currentProject?.baselineId,
1072    ]);
```

```
1073
1074    const handleCellEdit = (
1075      line_item_id: string,
1076      month: number,
1077      type: "forecast" | "actual"
1078    ) => {
1079      const cell = forecastData.find(
1080        (c) => c.line_item_id === line_item_id && c.month
     === month
1081      );
1082      setEditingCell({ line_item_id, month, type });
1083      const currentValue = type === "forecast" ?
     cell?.forecast : cell?.actual;
1084      setEditValue(currentValue?.toString() || "0");
1085    };
1086
1087    const handleCellSave = () => {
1088      if (editingCell) {
1089        let pendingChange: ForecastRow | null = null;
1090        const updatedData = forecastData.map((cell) => {
1091          if (
1092            cell.line_item_id === editingCell.line_item_id
     &&
1093            cell.month === editingCell.month
1094          ) {
1095            const newValue = parseFloat(editValue) || 0;
1096            const updates =
1097              editingCell.type === "forecast"
1098                ? { forecast: newValue, variance: newValue -
     cell.planned }
1099                : { actual: newValue };
1100
1101            const nextCell: ForecastRow = {
1102              ...cell,
1103              ...updates,
1104              last_updated: new Date().toISOString(),
1105              updated_by: user?.login || "current-user",
1106            };
1107
1108            // Track the pending change
```

```
875       setEditingCell({ line_item_id, month, type });
876       const currentValue = type === 'forecast' ? cell?.forecast : cell?.actual;
877       setEditValue(currentValue?.toString() || '0');
878     };
879
880     const handleCellSave = () => {
881       if (editingCell) {
882         let pendingChange: ForecastRow | null = null;
883         const updatedData = forecastData.map(cell => {
884           if (cell.line_item_id === editingCell.line_item_id && cell.month ===
     editingCell.month) {
885             const newValue = parseFloat(editValue) || 0;
886             const updates = editingCell.type === 'forecast'
887               ? { forecast: newValue, variance: newValue - cell.planned }
888               : { actual: newValue };
889
890             const nextCell: ForecastRow = {
891               ...cell,
892               ...updates,
893               last_updated: new Date().toISOString(),
894               updated_by: user?.login || 'current-user'
895             };
896
897             // Track the pending change
```

```
1109            if (editingCell.type === "actual") {
1110              pendingChange = nextCell;
1111            } else if (editingCell.type === "forecast") {
1112              pendingChange = nextCell;
1113            }
1114
1115            return nextCell;
1116          }
1117          return cell;
1118        });
1119        setForecastData(updatedData);
1120
1121        if (pendingChange) {
1122          const changeKey = `${pendingChange.projectId || selectedProjectId}-${
1123            pendingChange.line_item_id
1124          }-${pendingChange.month}`;
1125          if (editingCell.type === "actual") {
1126            setDirtyActuals((prev) => ({
1127              ...prev,
1128              [changeKey]: pendingChange as ForecastRow,
1129            }));
1130          } else if (editingCell.type === "forecast") {
1131            setDirtyForecasts((prev) => ({
1132              ...prev,
1133              [changeKey]: pendingChange as ForecastRow,
1134            }));
1135          }
1136        }
1137
1138        setEditingCell(null);
1139        toast.success(
1140          `${
1141            editingCell.type === "forecast" ? "Pronóstico" : "Real"
1142          } actualizado correctamente`
1143        );
1144      }
```

```
898            if (editingCell.type === 'actual') {
899              pendingChange = nextCell;
900            } else if (editingCell.type === 'forecast') {
901              pendingChange = nextCell;
902            }
903
904            return nextCell;
905          }
906          return cell;
907        });
908        setForecastData(updatedData);
909
910        if (pendingChange) {
911          const changeKey = `${pendingChange.projectId || selectedProjectId}-${pendingChange.line_item_id}-${pendingChange.month}`;
912          if (editingCell.type === 'actual') {
913            setDirtyActuals(prev => ({ ...prev, [changeKey]: pendingChange as ForecastRow }));
914          } else if (editingCell.type === 'forecast') {
915            setDirtyForecasts(prev => ({ ...prev, [changeKey]: pendingChange as ForecastRow }));
916          }
917        }
918
919        setEditingCell(null);
920        toast.success(`${editingCell.type === 'forecast' ? 'Pronóstico' : 'Real'} actualizado correctamente`);
921      }
```

```
1145      };
1146
1147      const handlePersistActuals = async () => {
1148        const entries = Object.values(dirtyActuals);
1149        if (entries.length === 0) {
1150          toast.info("No hay cambios de valores reales para guardar");
1151          return;
1152        }
1153
1154        setSavingActuals(true);
1155        try {
1156          const currentYear = new Date().getFullYear();
1157          const payload: PayrollActualInput[] = entries
1158            .map((cell) => {
1159              const projectId = cell.projectId || selectedProjectId;
1160              const matchedLineItem = lineItemsForGrid.find((item) => {
1161                const lineItemProjectId = (item as { projectId?: string })
1162                  .projectId;
1163                return (
1164                  item.id === cell.line_item_id &&
1165                  (!lineItemProjectId || lineItemProjectId === projectId)
1166                );
1167              });
1168              const monthKey = `${currentYear}-${String(cell.month).padStart(
1169                2,
1170                "0"
1171              )}`;
1172
1173              if (!projectId) return null;
1174              const currency = (matchedLineItem?.currency ||
1175                "USD") as PayrollActualInput["currency"];
1176
1177              return {
1178                projectId,
```

```
922       };
923
924       const handlePersistActuals = async () => {
925         const entries = Object.values(dirtyActuals);
926         if (entries.length === 0) {
927           toast.info('No hay cambios de valores reales para guardar');
928           return;
929         }
930
931         setSavingActuals(true);
932         try {
933           const currentYear = new Date().getFullYear();
934           const payload: PayrollActualInput[] = entries
935             .map(cell => {
936               const projectId = cell.projectId || selectedProjectId;
937               const matchedLineItem = lineItemsForGrid.find(item => {
938                 const lineItemProjectId = (item as { projectId?: string }).projectId;
939                 return item.id === cell.line_item_id && (!lineItemProjectId || lineItemProjectId === projectId);
940               });
941               const monthKey = `${currentYear}-${String(cell.month).padStart(2, '0')}`;
942
943               if (!projectId) return null;
944               const currency = (matchedLineItem?.currency || 'USD') as PayrollActualInput["currency"];
945
946               return {
947                 projectId,
```

```
1179                month: monthKey,
1180                rubroId: cell.line_item_id,
1181                amount: Number(cell.actual) || 0,
1182                currency,
1183                resourceCount: matchedLineItem?.qty
1184                  ? Number(matchedLineItem.qty)
1185                  : undefined,
1186                notes: cell.notes,
1187                uploadedBy: user?.email || user?.login,
1188                source: "sdmt-forecast",
1189              } as PayrollActualInput;
1190            })
1191            .filter((row): row is PayrollActualInput => Boolean(row));
1192
1193          if (payload.length === 0) {
1194            toast.error("No pudimos construir los datos para guardar en nómina.");
1195            return;
1196          }
1197
1198          await bulkUploadPayrollActuals(payload);
1199          toast.success("Valores reales enviados a Nómina (DynamoDB)");
1200          setDirtyActuals({});
1201        } catch (error) {
1202          console.error("❌ Error al guardar valores reales", error);
1203          const message = handleFinanzasApiError(error, {
1204            onAuthError: login,
1205            fallback: "No pudimos guardar los valores reales.",
1206          });
1207          setForecastError((prev) => prev || message);
1208        } finally {
1209          setSavingActuals(false);
1210        }
   @@ -1213,7 +980,7 @@ export function SDMTForecast() {
1213      const handlePersistForecasts = async () => {
1214        const entries: ForecastRow[] = Object.values(dirtyForecasts);
```

```
948                 month: monthKey,
949                 rubroId: cell.line_item_id,
950                 amount: Number(cell.actual) || 0,
951                 currency,
952                 resourceCount: matchedLineItem?.qty ? Number(matchedLineItem.qty) : undefined,
953                 notes: cell.notes,
954                 uploadedBy: user?.email || user?.login,
955                 source: 'sdmt-forecast',
956               } as PayrollActualInput;
957             })
958             .filter((row): row is PayrollActualInput => Boolean(row));
959
960           if (payload.length === 0) {
961             toast.error('No pudimos construir los datos para guardar en nómina.');
962             return;
963           }
964
965           await bulkUploadPayrollActuals(payload);
966           toast.success('Valores reales enviados a Nómina (DynamoDB)');
967           setDirtyActuals({});
968         } catch (error) {
969           console.error('❌ Error al guardar valores reales', error);
970           const message = handleFinanzasApiError(error, {
971             onAuthError: login,
972             fallback: 'No pudimos guardar los valores reales.',
973           });
974           setForecastError(prev => prev || message);
975         } finally {
976           setSavingActuals(false);
977         }
980       const handlePersistForecasts = async () => {
981         const entries: ForecastRow[] = Object.values(dirtyForecasts);
```

```
1215       if (entries.length === 0) {
1216         toast.info("No hay cambios de pronóstico para guardar");
1217         return;
1218       }
1219
     @@ -1226,46 +993,45 @@ export function SDMTForecast() {
1226       try {
1227         // Group by project for API calls
1228         const byProject = new Map<string, ForecastRow[]>();
1229         entries.forEach((cell) => {
1230           const projectId = cell.projectId || selectedProjectId;
1231           if (!projectId) return;
1232
1233           if (!byProject.has(projectId)) {
1234             byProject.set(projectId, []);
1235           }
1236           byProject.get(projectId)!.push(cell);
1237         });
1238
1239         // Send updates per project using bulkUpsertForecast with monthIndex format
1240         // The API expects: {items: [{rubroId, month: number, forecast}]}
1241         // We send monthIndex as a number, and the backend will compute the calendar month
1242         for (const [projectId, projectCells] of byProject.entries()) {
1243           const items = projectCells.map((cell) => {
1244             // Validate month is in valid range (up to baseline duration, fallback 60)
1245             const maxMonths = getBaselineDuration(baselineDetail);
1246             const monthIndex = Math.max(1, Math.min(maxMonths, cell.month));
1247             return {
1248               rubroId: cell.line_item_id,
1249               month: monthIndex,
1250               forecast: Number(cell.forecast) || 0,
1251             };
1252           });
1253
```

```
982       if (entries.length === 0) {
983         toast.info('No hay cambios de pronóstico para guardar');
984         return;
985       }
986
993       try {
994         // Group by project for API calls
995         const byProject = new Map<string, ForecastRow[]>();
996         entries.forEach(cell => {
997           const projectId = cell.projectId || selectedProjectId;
998           if (!projectId) return;
999
1000          if (!byProject.has(projectId)) {
1001            byProject.set(projectId, []);
1002          }
1003          byProject.get(projectId)!.push(cell);
1004        });
1005
1006        // Send updates per project using bulkUpsertForecast with monthIndex format
1007        // The API expects: {items: [{rubroId, month: number (1-12), forecast}]}
1008        // We send monthIndex as a number, and the backend will compute the calendar month
1009        for (const [projectId, projectCells] of byProject.entries()) {
1010          const items = projectCells.map(cell => {
1011            // Validate month is in valid range (1-12)
1012            const monthIndex = Math.max(1, Math.min(12, cell.month));
1013            return {
1014              rubroId: cell.line_item_id,
1015              month: monthIndex, // Send as numeric monthIndex (1-12)
1016              forecast: Number(cell.forecast) || 0,
1017            };
1018          });
1019
```

```
1254          await finanzasClient.bulkUpsertForecast(projectId, items);
1255        }
1256
1257        toast.success("Pronósticos ajustados guardados exitosamente");
1258        setDirtyForecasts({});
1259
1260        // Reload forecast data to show persisted values
1261        await loadForecastData();
1262      } catch (error) {
1263        console.error("❌ Error al guardar pronósticos", error);
1264        const message = handleFinanzasApiError(error, {
1265          onAuthError: login,
1266          fallback: "No pudimos guardar los pronósticos ajustados.",
1267        });
1268        setForecastError((prev) => prev || message);
1269      } finally {
1270        setSavingForecasts(false);
1271      }
•••   @@ -1289,17 +1055,15 @@ export function SDMTForecast() {
1289        setBudgetMissingYear(resolution.state.missingYear);
1290
1291        // If 404, it means no budget is set for this year - that's okay
1292        if (resolution.status === "missing") {
1293          console.warn(
1294            `[SDMTForecast] ⚠️ No annual budget configured for ${year}`
1295          );
1296          return;
1297        }
1298
1299        console.error("Error loading annual budget:", error);
1300        const message = handleFinanzasApiError(error, {
1301          onAuthError: login,
1302          fallback: "No pudimos cargar el presupuesto anual.",
1303        });
```

```
1020          await finanzasClient.bulkUpsertForecast(projectId, items);
1021        }
1022
1023        toast.success('Pronósticos ajustados guardados exitosamente');
1024        setDirtyForecasts({});
1025
1026        // Reload forecast data to show persisted values
1027        await loadForecastData();
1028      } catch (error) {
1029        console.error('❌ Error al guardar pronósticos', error);
1030        const message = handleFinanzasApiError(error, {
1031          onAuthError: login,
1032          fallback: 'No pudimos guardar los pronósticos ajustados.',
1033        });
1034        setForecastError(prev => prev || message);
1035      } finally {
1036        setSavingForecasts(false);
1037      }
1055        setBudgetMissingYear(resolution.state.missingYear);
1056
1057        // If 404, it means no budget is set for this year - that's okay
1058        if (resolution.status === 'missing') {
1059          console.warn(`[SDMTForecast] ⚠️ No annual budget configured for ${year}`);
1060          return;
1061        }
1062
1063        console.error('Error loading annual budget:', error);
1064        const message = handleFinanzasApiError(error, {
1065          onAuthError: login,
1066          fallback: 'No pudimos cargar el presupuesto anual.',
1067        });
```

```
1304          toast.error(message);
1305        } finally {
•••     @@ -1310,7 +1074,7 @@ export function SDMTForecast() {
1310      // Load Budget Overview for KPIs (only in portfolio view)
1311      const loadBudgetOverview = async (year: number) => {
1312        if (!isPortfolioView) return;
1313
1314        try {
1315          const overview = await finanzasClient.getAllInBudgetOverview(year);
1316          if (!overview) {
•••     @@ -1321,15 +1085,15 @@ export function SDMTForecast() {
1321            return;
1322          }
1323          setBudgetOverview(overview);
1324          console.log("[SDMTForecast] Budget overview loaded:", overview);
1325        } catch (error: any) {
1326          if (isBudgetNotFoundError(error)) {
1327            console.warn(`[SDMTForecast] ⚠️ Budget overview not found for ${year}`);
1328            setBudgetOverview(null);
1329            return;
1330          }
1331          // Don't show error to user, just log it - this is optional enhancement
1332          console.error("Error loading budget overview:", error);
1333          setBudgetOverview(null);
1334        }
1335      };
•••     @@ -1338,28 +1102,24 @@ export function SDMTForecast() {
1338      const handleSaveAnnualBudget = async () => {
1339        const amount = parseFloat(budgetAmount);
1340        if (isNaN(amount) || amount < 0) {
1341          toast.error("Por favor ingrese un monto válido");
1342          return;
1343        }
1344
1345        setSavingBudget(true);

1068          toast.error(message);
1069        } finally {
1074      // Load Budget Overview for KPIs (only in portfolio view)
1075      const loadBudgetOverview = async (year: number) => {
1076        if (!isPortfolioView) return;
1077
1078        try {
1079          const overview = await finanzasClient.getAllInBudgetOverview(year);
1080          if (!overview) {
1085            return;
1086          }
1087          setBudgetOverview(overview);
1088          console.log('[SDMTForecast] Budget overview loaded:', overview);
1089        } catch (error: any) {
1090          if (isBudgetNotFoundError(error)) {
1091            console.warn(`[SDMTForecast] ⚠️ Budget overview not found for ${year}`);
1092            setBudgetOverview(null);
1093            return;
1094          }
1095          // Don't show error to user, just log it - this is optional enhancement
1096          console.error('Error loading budget overview:', error);
1097          setBudgetOverview(null);
1098        }
1099      };
1102      const handleSaveAnnualBudget = async () => {
1103        const amount = parseFloat(budgetAmount);
1104        if (isNaN(amount) || amount < 0) {
1105          toast.error('Por favor ingrese un monto válido');
1106          return;
1107        }
1108
1109        setSavingBudget(true);
```

```
1346       try {
1347         const result = await finanzasClient.putAllInBudget(
1348           budgetYear,
1349           amount,
1350           budgetCurrency
1351         );
1352         setBudgetLastUpdated(result.updated_at);
1353         toast.success("Presupuesto anual guardado exitosamente");
1354
1355         // Reload budget and budget overview to update KPIs
1356         await loadAnnualBudget(budgetYear);
1357         await loadBudgetOverview(budgetYear);
1358       } catch (error) {
1359         console.error("Error saving annual budget:", error);
1360         const message = handleFinanzasApiError(error, {
1361           onAuthError: login,
1362           fallback: "No pudimos guardar el presupuesto anual.",
1363         });
1364         toast.error(message);
1365       } finally {
     @@ -1370,7 +1130,7 @@ export function SDMTForecast() {
1370     // Load Monthly Budget
1371     const loadMonthlyBudget = async (year: number) => {
1372       if (!isPortfolioView) return; // Monthly budgets only in portfolio view
1373
1374       setLoadingMonthlyBudget(true);
1375       try {
1376         const monthlyBudget = await finanzasClient.getAllInBudgetMonthly(year);
     @@ -1383,21 +1143,19 @@ export function SDMTForecast() {
1383         }
1384         if (monthlyBudget && monthlyBudget.months) {
1385           // Convert from API format (month: "YYYY-MM", amount) to internal format (month: 1-12, budget)
1386           const budgets: MonthlyBudgetInput[] = monthlyBudget.months
1387             .map((m) => {
```

```
1110       try {
1111         const result = await finanzasClient.putAllInBudget(budgetYear, amount, budgetCurrency);
1112         setBudgetLastUpdated(result.updated_at);
1113         toast.success('Presupuesto anual guardado exitosamente');
1114
1115         // Reload budget and budget overview to update KPIs
1116         await loadAnnualBudget(budgetYear);
1117         await loadBudgetOverview(budgetYear);
1118       } catch (error) {
1119         console.error('Error saving annual budget:', error);
1120         const message = handleFinanzasApiError(error, {
1121           onAuthError: login,
1122           fallback: 'No pudimos guardar el presupuesto anual.',
1123         });
1124         toast.error(message);
1125       } finally {
     @@ -1370,7 +1130,7 @@ export function SDMTForecast() {
1130     // Load Monthly Budget
1131     const loadMonthlyBudget = async (year: number) => {
1132       if (!isPortfolioView) return; // Monthly budgets only in portfolio view
1133
1134       setLoadingMonthlyBudget(true);
1135       try {
1136         const monthlyBudget = await finanzasClient.getAllInBudgetMonthly(year);
     @@ -1383,21 +1143,19 @@ export function SDMTForecast() {
1143         }
1144         if (monthlyBudget && monthlyBudget.months) {
1145           // Convert from API format (month: "YYYY-MM", amount) to internal format (month: 1-12, budget)
1146           const budgets: MonthlyBudgetInput[] = monthlyBudget.months.map(m => {
1147             const monthMatch = m.month.match(/^\d{4}-(\d{2})$/);
```

```
1388              const monthMatch = m.month.match(/^\d{4}-(\d{2})$/);
1389              const monthNum = monthMatch ? parseInt(monthMatch[1], 10) : 0;
1390              return {
1391                month: monthNum,
1392                budget: m.amount,
1393              };
1394            })
1395            .filter((b) => b.month >= 1 && b.month <= 60); // Support up to 60 months
1396
1397          setMonthlyBudgets(budgets);
1398          setMonthlyBudgetLastUpdated(monthlyBudget.updated_at || null);
1399          setMonthlyBudgetUpdatedBy(monthlyBudget.updated_by || null);
1400
1401          // If we have saved monthly budgets, enable the monthly budget mode
1402          if (budgets.length > 0) {
1403            setUseMonthlyBudget(true);
     @@ -1421,13 +1179,11 @@ export function SDMTForecast() {
1421          setMonthlyBudgetUpdatedBy(null);
1422          setUseMonthlyBudget(false);
1423        } else {
1424          console.error("Error loading monthly budget:", error);
1425
1426          // Show user-friendly error for network failures
1427          if (error instanceof TypeError && error.message.includes("fetch")) {
1428            toast.error(
1429              "Error de red al cargar presupuesto mensual. Verifique la conexión e intente nuevamente."
1430            );
1431          }
1432          // Don't show toast for other errors (optional feature, may not be configured)
1433        }
     @@ -1436,60 +1192,6 @@ export function SDMTForecast() {
1436      }
```

```
1148              const monthNum = monthMatch ? parseInt(monthMatch[1], 10) : 0;
1149              return {
1150                month: monthNum,
1151                budget: m.amount,
1152              };
1153            }).filter(b => b.month >= 1 && b.month <= 12);
1154
1155          setMonthlyBudgets(budgets);
1156          setMonthlyBudgetLastUpdated(monthlyBudget.updated_at || null);
1157          setMonthlyBudgetUpdatedBy(monthlyBudget.updated_by || null);
1158
1159          // If we have saved monthly budgets, enable the monthly budget mode
1160          if (budgets.length > 0) {
1161            setUseMonthlyBudget(true);
1179          setMonthlyBudgetUpdatedBy(null);
1180          setUseMonthlyBudget(false);
1181        } else {
1182          console.error('Error loading monthly budget:', error);
1183
1184          // Show user-friendly error for network failures
1185          if (error instanceof TypeError && error.message.includes('fetch')) {
1186            toast.error('Error de red al cargar presupuesto mensual. Verifique la conexión e intente nuevamente.');
1187          }
1188          // Don't show toast for other errors (optional feature, may not be configured)
1189        }
1192      }
```

```
1437      };
1438
1439      // Save Monthly Budget
1440      const handleSaveMonthlyBudget = async () => {
1441        if (monthlyBudgets.length === 0) {
1442          toast.error("Ingrese al menos un presupuesto mensual");
1443          return;
1444        }
1445
1446        setSavingMonthlyBudget(true);
1447        try {
1448          // Convert from internal format (month: 1-12, budget) to API format (month: "YYYY-MM", amount)
1449          const months = monthlyBudgets.map((mb) => ({
1450            month: `${budgetYear}-${String(mb.month).padStart(2, "0")}`,
1451            amount: mb.budget,
1452          }));
1453
1454          const result = await finanzasClient.putAllInBudgetMonthly(
1455            budgetYear,
1456            budgetCurrency,
1457            months
1458          );
1459          setMonthlyBudgetLastUpdated(result.updated_at);
1460          setMonthlyBudgetUpdatedBy(result.updated_by);
1461          toast.success("Presupuesto mensual guardado exitosamente");
1462
1463          // Reload monthly budget and budget overview to update KPIs and grid
1464          await loadMonthlyBudget(budgetYear);
1465          await loadBudgetOverview(budgetYear);
1466        } catch (error) {
1467          console.error("Error saving monthly budget:", error);
1468
1469          // Provide detailed error message for network failures
1470          let message: string;
```

```
1193      };
1194
```

```
1471        if (error instanceof TypeError &&
    error.message.includes("fetch")) {
1472          message =
1473            "Error de red al guardar presupuesto mensual.
    Verifique la conexión, configuración de CORS, y la URL
    base de la API en las variables de entorno.";
1474        } else {
1475          message = handleFinanzasApiError(error, {
1476            onAuthError: login,
1477            fallback: "No pudimos guardar el presupuesto
    mensual.",
1478          });
1479        }
1480        toast.error(message);
1481      } finally {
1482        setSavingMonthlyBudget(false);
1483      }
1484    };
1485
1486    // Reset monthly budget to auto-distribution
1487    const handleResetMonthlyBudget = () => {
1488      setMonthlyBudgets([]);
1489      setUseMonthlyBudget(false);
1490      toast.info("Presupuesto mensual restablecido a
    distribución automática");
1491    };
1492
1493    // Save budget from rubros table inline editing
1494    const handleSaveBudgetFromTable = async (
1495      budgets: Array<{ month: number; budget: number }>
```

```
1195    // Save budget from rubros table inline editing
1196    const handleSaveBudgetFromTable = async (
1197      budgets: Array<{ month: number; budget: number }>
```

...