

CURRENT

PRESENT

financial-planning-u / src / features / sdmr / 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
54   "@/components/ChartInsightsPanel";
55 import LineChartComponent from
56   "@/components/charts/LineChart";
57 import { StackedColumnsChart } from
58   "@/components/charts/StackedColumnsChart";
59 import ModuleBadge from "@/components/ModuleBadge";
60 import LoadingSpinner from "@/components>LoadingSpinner";
61 import type { ForecastCell, LineItem } from
62   "@/types/domain";
63 import { useAuth } from "@/hooks/useAuth";
64 import { ALL_PROJECTS_ID, useProject } from
65   "@/contexts/ProjectContext";
66
67
68
69
70
71 import {
72   Dialog,
73   DialogContent,
74   DialogDescription,
75   DialogHeader,
76   DialogTitle,
77   DialogTrigger,
78 } from '@/components/ui/dialog';
79 import { Collapsible, CollapsibleContent, CollapsibleTrigger } from
80   '@/components/ui/collapsible';
81 import { Share2, TrendingUp, TrendingDown, ExternalLink, FileSpreadsheet, Info,
82   Calculator, ChevronDown } from 'lucide-react';
83 import { toast } from 'sonner';
84 import { ChartInsightsPanel } from '@/components/ChartInsightsPanel';
85 import LineChartComponent from '@/components/charts/LineChart';
86 import { StackedColumnsChart } from '@/components/charts/StackedColumnsChart';
87 import ModuleBadge from '@/components/ModuleBadge';
88 import LoadingSpinner from '@/components>LoadingSpinner';
89 import type { ForecastCell, LineItem } from '@/types/domain';
90 import { useAuth } from '@/hooks/useAuth';
91 import { ALL_PROJECTS_ID, useProject } from '@/contexts/ProjectContext';
92 import { handleFinanzasApiError } from
93   '@/features/sdmt/cost/utils/errorHandling';
94 import { useNavigate, useLocation } from 'react-router-dom';
95 import { excelExporter, downloadExcelFile } from '@/lib/excel-export';
96 import { PDFExporter, formatReportCurrency, formatReportPercentage,
97   getChangeType } from '@/lib/pdf-export';
98 import { computeTotals, computeVariance } from '@/lib/forecast/analytics';
99 import { normalizeForecastCells } from
100   '@/features/sdmt/cost/utils/dataAdapters';
101 import { useProjectLineItems } from '@/hooks/useProjectLineItems';
102 import { bulkUploadPayrollActuals, type PayrollActualInput, getProjectRubros,
103   getBaselineById, type BaselineDetail } from '@/api/finanzas';
104 import { getForecastPayload, getProjectInvoices } from './forecastService';
105 import finanzasClient, { type BaselineDetailResponse } from
106   '@/api/finanzasClient';
107 import { ES_TEXTS } from '@/lib/i18n/es';
108 import { BaselineStatusPanel } from
109   '@/components/baseline/BaselineStatusPanel';
110 import { BudgetSimulatorCard } from './BudgetSimulatorCard';
111 import { MonthlyBudgetCard } from './MonthlyBudgetCard';
```

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

```
54 import { PortfolioSummaryView } from './PortfolioSummaryView';
55
56 import { ForecastSummaryBar } from './components/ForecastSummaryBar';
57
58 import { ForecastChartsPanel } from './components/ForecastChartsPanel';
59 import { ForecastRubrosTable } from './components/ForecastRubrosTable';
60 import { TopVarianceProjectsTable } from
61   './components/TopVarianceProjectsTable';
62 import { TopVarianceRubrosTable } from './components/TopVarianceRubrosTable';
63 import { MonthlySnapshotGrid } from './components/MonthlySnapshotGrid';
64 import { DataHealthPanel } from '@/components/finanzas/DataHealthPanel';
65 import type { BudgetSimulationState, SimulatedMetrics } from
66   './budgetSimulation';
67 import { applyBudgetSimulation, applyBudgetToTrends } from
68   './budgetSimulation';
69 import { isBudgetNotFoundError, resolveAnnualBudgetState } from
70   './budgetState';
71
72 import {
73   allocateBudgetMonthly,
74   aggregateMonthlyTotals,
```

```
91 import { TopVarianceProjectsTable } from
92   './components/TopVarianceProjectsTable';
93 import { TopVarianceRubrosTable } from
94   './components/TopVarianceRubrosTable';
95 import { MonthlySnapshotGrid } from
96   './components/MonthlySnapshotGrid';
97 import { DataHealthPanel } from
98   '@/components/finanzas/DataHealthPanel';
99 import type {
100   BudgetSimulationState,
101   SimulatedMetrics,
102 } from './budgetSimulation';
103 import { applyBudgetSimulation, applyBudgetToTrends } from
104   './budgetSimulation';
105 import { isBudgetNotFoundError, resolveAnnualBudgetState } from
106   './budgetState';
107 import {
108   allocateBudgetMonthly,
109   aggregateMonthlyTotals,
110   allocateBudgetWithMonthlyInputs,
111   calculateRunwayMetrics,
112   type MonthlyAllocation,
113   type MonthlyBudgetInput,
114   type RunwayMetrics,
115 } from './budgetAllocation';
116 import {
117   buildCategoryTotals,
118   buildCategoryRubros,
119   buildPortfolioTotals,
120   import { buildProjectTotals, buildProjectRubros } from
121     './projectGrouping';
122   import { getTaxonomyById } from '@/lib/rubros/canonical-
123     taxonomy';
124   import { clampMonthIndex } from
125     './monthHelpers';
126   // ---- Exported helpers for month support and testing ---
127   -
128   // Re-exported from monthHelpers.ts for backward
129   compatibility
130   export { getBaselineDuration, clampMonthIndex };
```

```
68   allocateBudgetWithMonthlyInputs,
69   calculateRunwayMetrics,
70   type MonthlyAllocation,
71   type MonthlyBudgetInput,
72   type RunwayMetrics,
73 } from './budgetAllocation';
74 import {
75   buildCategoryTotals,
76   buildCategoryRubros,
77   buildPortfolioTotals,
78 } from './categoryGrouping';
79 import { buildProjectTotals, buildProjectRubros } from './projectGrouping';
80 import { rubrosFromAllocations } from '@/features/sdmt/utils';
```

```
123 // -----
124 --
125 // TODO: Backend Integration for Change Request Impact on
126 // Forecast
127 // When a change request is approved in SDMTChanges, the
128 // backend should:
129 ...
130     @@ -173,134 +130,63 @@ type LineItemLike = Record<string, unknown>;
131
132 // Constants
133 const MINIMUM_PROJECTS_FOR_PORTFOLIO = 2; // ALL_PROJECTS
134 + at least one real project
135
136
137 // Feature flags for new forecast layout
138 const NEW_FORECAST_LAYOUT_ENABLED =
139 import.meta.env.VITE_FINZ_NEW_FORECAST_LAYOUT === 'true';
140 const SHOW_KEY_TRENDS =
141 import.meta.env.VITE_FINZ_SHOW_KEYTRENDS === 'true';
142
143 // Backward compatibility: HIDE_KEY_TRENDS is deprecated,
144 // use SHOW_KEY_TRENDS instead
145
146 // Feature flag to hide executive key-trends (projects &
147 // rubros) cards
148 const HIDE_KEY_TRENDS =
149 import.meta.env.VITE_FINZ_HIDE_KEY_TRENDS === 'true';
150
151 // Feature flag to hide Real Annual Budget KPIs in
152 // TODOS/Portfolio view
153 const HIDE_REAL_ANNUAL_KPIS =
154 import.meta.env.VITE_FINZ_HIDE_REAL_ANNUAL_KPIS ===
155 'true';
156
157 // Feature flag to hide Resumen de Portafolio in TODOS
158 // mode
159 const HIDE_PROJECT_SUMMARY =
160 import.meta.env.VITE_FINZ_HIDE_PROJECT_SUMMARY === 'true';
161
162 // Feature flag to show Portfolio KPI tiles in new layout
163 // (default false for minimal view)
164 const SHOW_PORTFOLIO_KPIS =
165 import.meta.env.VITE_FINZ_SHOW_PORTFOLIO_KPIS === 'true';
166
167
168 // Debug logging for feature flags (development only)
169 if (import.meta.env.DEV) {
170     console.log('[SDMTForecast] Feature Flags:', {
171         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,
208 HIDE_EXPANDABLE_PROJECT_LIST,
209 // and HIDE_RUNWAY_METRICS are declared and used inside
PortfolioSummaryView.tsx)
210
211 export function SDMTForecast() {
212   const [forecastData, setForecastData] =
useState<ForecastRow[]>([]);
213   const [loading, setLoading] = useState(true);
214   const [isLoadingForecast, setIsLoadingForecast] =
useState(true);
215   const [forecastError, setForecastError] =
useState<string | null>(null);
216   const [exporting, setExporting] = useState<"excel" |
"pdf" | null>(null);
217   const [editingCell, setEditingCell] = useState<{
218     line_item_id: string;
219     month: number;
220     type: "forecast" | "actual";
221   } | null>(null);
222   const [editValue, setEditValue] = useState("");
223   const [dataSource, setDataSource] = useState<"api" |
"mock">("api");
224   const [generatedAt, setGeneratedAt] = useState<string | null>(null);
225   const [portfolioLineItems, setPortfolioLineItems] =
useState<
226     ProjectLineItem[]
227   >([]);
228   const [dirtyActuals, setDirtyActuals] =
useState<Record<string, ForecastRow>>(
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 <code>const [savingActuals, setSavingActuals] = useState(false);</code>	145 <code>const [savingActuals, setSavingActuals] = useState(false);</code>
231 <code>const [dirtyForecasts, setDirtyForecasts] = useState< Record<string, ForecastRow> >({});</code>	146 <code>const [dirtyForecasts, setDirtyForecasts] = useState<Record<string, ForecastRow>>({});</code>
232 <code>Record<string, ForecastRow></code>	
233 <code>>({});</code>	
234 <code>const [savingForecasts, setSavingForecasts] = useState(false);</code>	147 <code>const [savingForecasts, setSavingForecasts] = useState(false);</code>
235 <code>// Baseline detail for FTE calculation</code>	148 <code>// Baseline detail and rubros source tracking</code>
236 <code>const [baselineDetail, setBaselineDetail] =</code>	149 <code>const [baselineDetail, setBaselineDetail] = useState<BaselineDetailResponse null>(null);</code>
237 <code>useState<BaselineDetail null>(null);</code>	150 <code>const [rubrosSource, setRubrosSource] = useState<'api' 'fallback' null> (null);</code>
238 <code>useState<BaselineDetail null>(null);</code>	151 <code>// Request ID tracking to prevent race conditions</code>
239	152 <code>const requestIdRef = useRef(0);</code>
	153 <code>// Request ID tracking to prevent race conditions</code>
240 <code>// Sorting state for forecast grid</code>	154 <code>const requestIdRef = useRef(0);</code>
241 <code>const [sortDirection, setSortDirection] = useState<"asc" "desc">("asc");</code>	155
242	156 <code>// Sorting state for forecast grid</code>
243 <code>// State for controlling rubros grid collapsible (TODOS view)</code>	157 <code>const [sortDirection, setSortDirection] = useState<'asc' 'desc'>('asc');</code>
244 <code>const [isRubrosGridOpen, setIsRubrosGridOpen] = useState(false);</code>	158
245 <code>// Breakdown view mode for TODOS/portfolio view (Proyectos vs Rubros)</code>	159 <code>// State for controlling rubros grid collapsible (TODOS view)</code>
246 <code>// The ForecastRubrosTable component has its own internal viewType that switches between</code>	160 <code>const [isRubrosGridOpen, setIsRubrosGridOpen] = useState(false);</code>
247 <code>// 'category' and 'project' views. This state tracks the user's preference at the page level.</code>	
248 <code>const [breakdownMode, setBreakdownMode] = useState<'project' 'rubros'>(() => {</code>	
249 <code> const stored = sessionStorage.getItem('forecastBreakdownMode');</code>	
250 <code> return stored === 'rubros' ? 'rubros' : 'project';</code>	
251 <code>});</code>	
252	161
253	
254 <code>// Handler for breakdown mode changes</code>	
255 <code>const handleBreakdownModeChange = (newMode: 'project' 'rubros') => {</code>	
256 <code> setBreakdownMode(newMode);</code>	
257 <code> sessionStorage.setItem('forecastBreakdownMode', newMode);</code>	

258 // Note: The ForecastRubrosTable component manages its 259 own viewMode internally 260 // and persists it in sessionStorage. This handler 261 updates the page-level state 262 // which could be used to control other aspects of the 263 view in the future. 264 }; 265 266 // Stale response guard: Track latest request to prevent 267 race conditions	
264 const latestRequestKeyRef = useRef<string>(“”); 265 const abortControllerRef = useRef<AbortController 266 null>(null); 267 // Ref for scrolling to rubros section when category is 268 // clicked 269 const rubrosSectionRef = useRef<HTMLDivElement>(null);	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);
270 const [budgetSimulation, setBudgetSimulation] = 271 useState<BudgetSimulationState>({ 272 enabled: false, 273 budgetTotal: “”, 274 factor: 1.0, 275 estimatedOverride: “”, 276 }); 277 // Annual Budget state	168 const [budgetSimulation, setBudgetSimulation] = 169 useState<BudgetSimulationState>({ 170 enabled: false, 171 budgetTotal: “”, 172 factor: 1.0, 173 estimatedOverride: “”, 174 });
278 const [budgetYear, setBudgetYear] = useState<number>(279 new Date().getFullYear() 280); 281 const [budgetAmount, setBudgetAmount] = useState<string>(282 “”); 283 const [budgetCurrency, setBudgetCurrency] = 284 useState<string>(“USD”); 285 const [budgetLastUpdated, setBudgetLastUpdated] = 286 useState<string null>(287 null 288); 289 const [loadingBudget, setLoadingBudget] = 290 useState(false);	175 // Annual Budget state 176 const [budgetYear, setBudgetYear] = useState<number>(new 177 Date().getFullYear()); 178 const [budgetAmount, setBudgetAmount] = useState<string>(“”); 179 const [budgetCurrency, setBudgetCurrency] = useState<string>(“USD”); 180 const [budgetLastUpdated, setBudgetLastUpdated] = useState<string null>(181 null); 182 const [budgetMissingYear, setBudgetMissingYear] = useState<number null>(183 null); 184 const [loadingBudget, setLoadingBudget] = useState(false);

290	const [savingBudget, setSavingBudget] = useState(false);	182	const [savingBudget, setSavingBudget] = useState(false);
291	// Monthly Budget state (new - per user request)	183	// Monthly Budget state (new - per user request)
292	const [monthlyBudgets, setMonthlyBudgets] = useState<MonthlyBudgetInput[]>([]);	184	const [monthlyBudgets, setMonthlyBudgets] = useState<MonthlyBudgetInput[]>([]);
293			
294);		
295	const [useMonthlyBudget, setUseMonthlyBudget] = useState(false);	185	const [useMonthlyBudget, setUseMonthlyBudget] = useState(false);
296	const [loadingMonthlyBudget, setLoadingMonthlyBudget] = useState(false);	186	const [loadingMonthlyBudget, setLoadingMonthlyBudget] = useState(false);
297	const [savingMonthlyBudget, setSavingMonthlyBudget] = useState(false);	187	const [savingMonthlyBudget, setSavingMonthlyBudget] = useState(false);
298	const [monthlyBudgetLastUpdated, setMonthlyBudgetLastUpdated] = useState<string null>(null);	188	const [monthlyBudgetLastUpdated, setMonthlyBudgetLastUpdated] = useState<string null>(null);
299	const [monthlyBudgetUpdatedBy, setMonthlyBudgetUpdatedBy] = useState<string null>(null);	189	const [monthlyBudgetUpdatedBy, setMonthlyBudgetUpdatedBy] = useState<string null>(null);
300			
301			
302			
303			
304	// Budget Overview state for KPIs	190	// Budget Overview state for KPIs
305	const [budgetOverview, setBudgetOverview] = useState<{	191	const [budgetOverview, setBudgetOverview] = useState<{
306	year: number;	192	year: number;
...	@@ -316,152 +202,140 @@ export function SDMTForecast() {		
316	};	202	};
317	} null>(null);	203	} null>(null);
318	const { user, login } = useAuth();	204	const { user, login } = useAuth();
319	const {	205	const { selectedProjectId, setSelectedProjectId, selectedPeriod, currentProject, projectChangeCount, projects } = useProject();
320	selectedProjectId,		
321	setSelectedProjectId,		
322	selectedPeriod,		
323	currentProject,		
324	projectChangeCount,		
325	projects,		
326	} = useProject();		
327	const navigate = useNavigate();	206	const navigate = useNavigate();
328	const location = useLocation();	207	const location = useLocation();
329	const {	208	const {
330	lineItems,	209	lineItems,
331	taxonomyByRubroid,		

332 isLoading: isLineItemsLoading,	210 isLoading: isLineItemsLoading,
333 error: lineItemsError,	211 error: lineItemsError,
334 } = useProjectLineItems({	212 } = useProjectLineItems({ useFallback: true, baselineId: currentProject?.baselineId });
335 useFallback: true,	
336 baselineId: currentProject?.baselineId,	
337 withTaxonomy: true,	
338 });	
339 const safeLineItems = useMemo(213 const safeLineItems = useMemo(
340 () => (Array.isArray(lineItems) ? lineItems : []),	214 () => (Array.isArray(lineItems) ? lineItems : []),
341 [lineItems]	215 [lineItems]
342);	216);
343 const isPortfolioView = selectedProjectId === ALL_PROJECTS_ID;	217 const isPortfolioView = selectedProjectId === ALL_PROJECTS_ID;
344 const forecastMode: "single" "all-projects" = isPortfolioView ? "all-projects" : "single";	218 const forecastMode: 'single' 'all-projects' = isPortfolioView ? 'all-projects' : 'single';
345 const lineItemsForGrid = isPortfolioView ? portfolioLineItems : safeLineItems;	219 const lineItemsForGrid = isPortfolioView ? portfolioLineItems : safeLineItems;
346 const projectStartDate = (currentProject as { start_date?: string } null) ?.start_date;	220 const projectStartDate = (currentProject as { start_date?: string } null)?.start_date;
347	221
348	222 // Helper function to get current month index (1-12) based on today's date and project start
349	223 const getCurrentMonthIndex = (): number => {
350	224 const today = new Date();
351 // Helper function to get current month index (1-12) based on today's date and project start	225 const currentYear = today.getFullYear();
352 const getCurrentMonthIndex = (): number => {	226 const currentMonth = today.getMonth() + 1; // 1-12
353 const today = new Date();	227
354 const currentYear = today.getFullYear();	228 // For portfolio view or when no project start date, use calendar month (1-12)
355 const currentMonth = today.getMonth() + 1; // 1-12	229 if (!projectStartDate isPortfolioView) {
356	230 return currentMonth;
357 // For portfolio view or when no project start date, use calendar month (1-12)	231 }
358 if (!projectStartDate isPortfolioView) {	232
359 return currentMonth;	233 // Calculate month index relative to project start date
360 }	234 const startDate = new Date(projectStartDate);
361	235 const startYear = startDate.getFullYear();
362 // Calculate month index relative to project start date	236 const startMonth = startDate.getMonth() + 1; // 1-12
363 const startDate = new Date(projectStartDate);	
364 const startYear = startDate.getFullYear();	
365 const startMonth = startDate.getMonth() + 1; // 1-12	

366		237	
367 // Calculate months elapsed since project start		238 // Calculate months elapsed since project start	
368 const monthsElapsed =		239 const monthsElapsed = (currentYear - startYear) * 12 + (currentMonth - startMonth) + 1;	
369 (currentYear - startYear) * 12 + (currentMonth - startMonth) + 1;		240	
370		241 // Clamp to 1-12 range (project month index)	
371 // Clamp according to baseline duration (fallback to 60)		242 return Math.max(1, Math.min(12, monthsElapsed));	
372 return clampMonthIndex(monthsElapsed, baselineDetail);		243 };	
373 };		244	
374		245 // Helper function to compute calendar month from monthIndex and project start date	
375 // Helper function to compute calendar month from monthIndex and project start date		246 const getCalendarMonth = (monthIndex: number): string =>	
376 const getCalendarMonth = (monthIndex: number): string =>		247 if (!projectStartDate) {	
{		248 // Fallback: display just the month name without year for consistency	
377 if (!projectStartDate) {		249 const monthNames = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'];	
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}`;		250 return monthNames[monthIndex - 1] `M\${monthIndex}`;	
394 }		251 }	
395		252	
396 const startDate = new Date(projectStartDate);		253 const startDate = new Date(projectStartDate);	
397 startDate.setUTCMonth(startDate.getUTCMonth() + (monthIndex - 1));		254 startDate.setUTCMonth(startDate.getUTCMonth() + (monthIndex - 1));	
398 const year = startDate.getUTCFullYear();		255 const year = startDate.getUTCFullYear();	
399 const month = startDate.getUTCMonth() + 1;		256 const month = startDate.getUTCMonth() + 1;	

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

```
281     if (abortControllerRef.current) {
282         abortControllerRef.current.abort();
283     }
284 };
285 }, [selectedProjectId, selectedPeriod, projectChangeCount,
286 currentProject?.baselineId]);
287
288 // Reload data when returning from reconciliation with refresh parameter
289 useEffect(() => {
290     const urlParams = new URLSearchParams(location.search);
291     const refreshParam = urlParams.get('_refresh');
292     if (refreshParam && selectedProjectId) {
293         if (import.meta.env.DEV) {
294             console.log('⌚ Forecast: Refreshing after reconciliation');
295         }
296         loadForecastData();
297     }
298 }, [location.search]);
299
300 // Load baseline details for FTE calculation
301 useEffect(() => {
302     if (!lineItemsError) return;
303     const message = handleFinanzasApiError(lineItemsError, {
304         onAuthError: login,
305         fallback: 'No se pudieron cargar los rubros para forecast.',
306     });
307     setForecastError((prev) => prev || message);
308 }, [lineItemsError, login]);
309
310 // Load baseline details for FTE calculation
311 useEffect(() => {
312     if (currentProject?.baselineId && !isPortfolioView) {
313         getBaselineById(currentProject.baselineId)
314             .then((data) => {
315                 if (import.meta.env.DEV) {
316                     console.log(
317                         '[SDMTForecast] Baseline details loaded for FTE
318 calculation'
319                     );
320                 }
321             })
322     }
323 }, [isPortfolioView]);
324
325 // Load baseline details for FTE calculation
326 useEffect(() => {
327     if (currentProject?.baselineId && !isPortfolioView) {
328         getBaselineById(currentProject.baselineId)
329             .then((data) => {
330                 if (import.meta.env.DEV) {
331                     console.log(
332                         '[SDMTForecast] Baseline details loaded for FTE
333 calculation'
334                     );
335                 }
336             })
337     }
338 }, [isPortfolioView]);
```

```

437         );
438     }
439     setBaselineDetail(data);
440   })
441   .catch((err) => {
442     console.error(`Failed to load baseline
443       details: ${err}`);
444     setBaselineDetail(null);
445   } else {
446     setBaselineDetail(null);
447   }
448 }, [currentProject?.baselineId, isPortfolioView]);
449
450 const loadForecastData = useCallback(async () => {
451   // If we're in portfolio (TODOS) view, we MUST load
452   // portfolio-wide forecast even when there is
453   // no selectedProjectId. Only skip forecast load when
454   // not in portfolio view and no project selected.
455   if (!isPortfolioView && !selectedProjectId) {
456     console.log(`✖ No project selected and not
457       portfolio view, skipping forecast load`);
458
459     return;
460   }
461
462   // Create a new abort controller for this request
463   abortControllerRef.current = new AbortController();
464
465   // Generate unique request key to identify this
466   // specific request
467   const requestKey = `${selectedProjectId}__${
468     currentProject?.baselineId || ""
469   }__${Date.now()}`;
470
471   latestRequestKeyRef.current = requestKey;
472
473   try {
...
474     @@@ -470,14 +344,14 @@ export function SDMTForecast() {
475       setForecastError(null);
476       setDirtyActuals({});
477       setDirtyForecasts({});
478     }
479   }
480   .catch((err) => {
481     console.error(`Failed to load baseline details: ${err}`);
482     setBaselineDetail(null);
483   } else {
484     setBaselineDetail(null);
485   }
486 }, [currentProject?.baselineId, isPortfolioView]);
487
488 const loadForecastData = async () => {
489   if (!selectedProjectId) {
490     console.log(`✖ No project selected, skipping forecast load`);
491
492     return;
493   }
494
495   // Create a new abort controller for this request
496   abortControllerRef.current = new AbortController();
497
498   // Generate unique request key to identify this specific request
499   const requestKey = `${selectedProjectId}__${currentProject?.baselineId || ""
500   }__${Date.now()}`;
501
502   latestRequestKeyRef.current = requestKey;
503
504   try {
505     @@@ -470,14 +344,14 @@ export function SDMTForecast() {
506       setForecastError(null);
507       setDirtyActuals({});
508       setDirtyForecasts({});
509     }
510   }
511   .catch((err) => {
512     console.error(`Failed to load baseline details: ${err}`);
513     setBaselineDetail(null);
514   } else {
515     setBaselineDetail(null);
516   }
517 }, [currentProject?.baselineId, isPortfolioView]);
518
519

```

473		347	
474	// Handle CURRENT_MONTH period - always load 12 months but filter to current month later	348	// Handle CURRENT_MONTH period - always load 12 months but filter to current month later
475	<code>const isCurrentMonthMode = selectedPeriod === "CURRENT_MONTH";</code>	349	<code>const isCurrentMonthMode = selectedPeriod === 'CURRENT_MONTH';</code>
476	<code>const months = isCurrentMonthMode ? 12 : parseInt(selectedPeriod);</code>	350	<code>const months = isCurrentMonthMode ? 12 : parseInt(selectedPeriod);</code>
477		351	
478	<code>if (import.meta.env.DEV) {</code>	352	<code>if (import.meta.env.DEV) {</code>
479	<code> console.debug("[Forecast] Loading data", {</code>	353	<code> console.debug('[Forecast] Loading data', {</code>
480	<code> projectId: selectedProjectId,</code>	354	<code> projectId: selectedProjectId,</code>
481	<code> months,</code>	355	<code> months,</code>
482	<code> isCurrentMonthMode,</code>	356	<code> isCurrentMonthMode,</code>
483	<code> selectedPeriod,</code>	357	<code> selectedPeriod,</code>
...	<code>@@ -488,38 +362,39 @@ export function SDMTForecast() {</code>		
488	<code> if (isPortfolioView) {</code>	362	<code> if (isPortfolioView) {</code>
489	<code> await loadPortfolioForecast(months, requestKey);</code>	363	<code> await loadPortfolioForecast(months, requestKey);</code>
490	<code>} else {</code>	364	<code>} else {</code>
		365	<code> // Load baseline rubros for single project (non-portfolio) view</code>
		366	<code> if (currentProject?.baselineId) {</code>
		367	<code> await loadBaselineRubros</code>
		368	<code> selectedProjectId,</code>
		369	<code> currentProject.baselineId,</code>
		370	<code> abortControllerRef.current?.signal</code>
		371	<code>);</code>
		372	<code>}</code>
491	<code> await loadSingleProjectForecast(selectedProjectId,</code>	373	<code> await loadSingleProjectForecast(selectedProjectId, months, requestKey);</code>
492	<code> months, requestKey);</code>	374	<code>}</code>
493		375	
494	<code> // Verify this is still the latest request before applying results</code>	376	<code> // Verify this is still the latest request before applying results</code>
495	<code> if (latestRequestKeyRef.current !== requestKey) {</code>	377	<code> if (latestRequestKeyRef.current !== requestKey) {</code>
496	<code> if (import.meta.env.DEV) {</code>	378	<code> if (import.meta.env.DEV) {</code>
497	<code> console.debug("[Forecast] Discarding stale response", {</code>	379	<code> console.debug('[Forecast] Discarding stale response', { requestKey,</code>
498	<code> requestKey,</code>		<code> latest: latestRequestKeyRef.current });</code>
499	<code> latest: latestRequestKeyRef.current,</code>		
500	<code> });</code>		
501	<code>}</code>	380	<code>}</code>
502	<code>return; // Stale response, ignore it</code>	381	<code>return; // Stale response, ignore it</code>

```

503      }
504    } catch (error) {
505      // Ignore aborted requests
506      if (error instanceof Error && error.name ===
507          "AbortError") {
508        if (import.meta.env.DEV) {
509          console.debug("[Forecast] Request aborted", {
510            requestKey });
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
524        // request
525        if (latestRequestKeyRef.current === requestKey) {
526          setForecastError(message);
527        }
528      }
529    }
530  }
531
532  setisLoadingForecast(false);
533}
534}
535}, [isPortfolioView, selectedProjectId,
536 currentProject?.baselineId, selectedPeriod, login]);
537const transformLineItemsToForecast =
538  lineItems: LineItemLike[],
539
540  @@ -549,9 +424,7 @@ export function SDMTForecast() {
541    return Number.isFinite(numeric) ? numeric :
542      fallback;
543  };
544
545  }
546
547  if (error instanceof Error && error.name === 'AbortError') {
548    if (import.meta.env.DEV) {
549      console.debug("[Forecast] Request aborted", {
550        requestKey });
551    }
552
553    console.error('✖ Failed to load forecast data for project:',
554      selectedProjectId, error);
555
556    const message = handleFinanzasApiError(error, {
557      onAuthError: login,
558      fallback: 'No se pudo cargar el forecast.',
559    });
560
561    // Only set error if this is still the latest request
562    if (latestRequestKeyRef.current === requestKey) {
563      setForecastError(message);
564    }
565  }
566
567  setisLoadingForecast(false);
568}
569
570  }
571
572  const transformLineItemsToForecast =
573    lineItems: LineItemLike[],
574
575  @@ -532,7 +407,7 @@ export function SDMTForecast() {
576    setisLoadingForecast(false);
577  }
578}
579
580  }
581
582  setisLoadingForecast(false);
583}
584
585  }
586
587  const transformLineItemsToForecast =
588    lineItems: LineItemLike[],
589
590  @@ -549,9 +424,7 @@ export function SDMTForecast() {
591    return Number.isFinite(numeric) ? numeric :
592      fallback;
593  };
594
595  }
596
597  return Number.isFinite(numeric) ? numeric : fallback;
598
599  }
600
601  }
602
603  }
604
605  }
606
607  }
608
609  }
610
611  }
612
613  }
614
615  }
616
617  }
618
619  }
620
621  }
622
623  }
624
625  }
626
627  }
628
629  }
630
631  }
632
633  }
634
635  }
636
637  }
638
639  }
640
641  }
642
643  }
644
645  }
646
647  }
648
649  }
650
651  }
652
653  }
654
655  }
656
657  }
658
659  }
660
661  }
662
663  }
664
665  }
666
667  }
668
669  }
670
671  }
672
673  }
674
675  }
676
677  }
678
679  }
680
681  }
682
683  }
684
685  }
686
687  }
688
689  }
690
691  }
692
693  }
694
695  }
696
697  }
698
699  }
699
700  }
701
702  }
703
704  }
705
706  }
707
708  }
709
710  }
711
712  }
713
714  }
715
716  }
717
718  }
719
720  }
721
722  }
723
724  }
725
726  }
727
728  }
729
730  }
731
732  }
733
734  }
735
736  }
737
738  }
739
740  }
741
742  }
743
744  }
745
746  }
747
748  }
749
750  }
751
752  }
753
754  }
755
756  }
757
758  }
759
759
760  }
761
762  }
763
764  }
765
766  }
767
768  }
769
769
770  }
771
772  }
773
774  }
775
776  }
777
778  }
779
779
780  }
781
782  }
783
784  }
785
786  }
787
787
788  }
789
790  }
791
792  }
793
794  }
795
796  }
797
797
798  }
799
800  }
801
802  }
803
804  }
805
806  }
807
808  }
809
809
810  }
811
812  }
813
814  }
815
816  }
817
817
818  }
819
820  }
821
822  }
823
824  }
825
826  }
827
827
828  }
829
830  }
831
832  }
833
834  }
835
836  }
837
837
838  }
839
840  }
841
842  }
843
844  }
845
846  }
847
847
848  }
849
850  }
851
852  }
853
854  }
855
856  }
857
858  }
859
859
860  }
861
862  }
863
864  }
865
866  }
867
867
868  }
869
870  }
871
872  }
873
874  }
875
876  }
877
877
878  }
879
880  }
881
882  }
883
884  }
885
886  }
887
887
888  }
889
890  }
891
892  }
893
894  }
895
896  }
897
897
898  }
899
900  }
901
902  }
903
904  }
905
906  }
907
908  }
909
909
910  }
911
912  }
913
914  }
915
916  }
917
917
918  }
919
920  }
921
922  }
923
924  }
925
926  }
927
927
928  }
929
930  }
931
932  }
933
934  }
935
936  }
937
937
938  }
939
940  }
941
942  }
943
944  }
945
946  }
947
947
948  }
949
950  }
951
952  }
953
954  }
955
956  }
957
957
958  }
959
960  }
961
962  }
963
964  }
965
966  }
967
967
968  }
969
970  }
971
972  }
973
974  }
975
976  }
977
977
978  }
979
980  }
981
982  }
983
984  }
985
986  }
987
987
988  }
989
990  }
991
992  }
993
994  }
995
996  }
997
997
998  }
999
1000  }
1001
1002  }
1003
1004  }
1005
1006  }
1007
1007
1008  }
1009
1010  }
1011
1012  }
1013
1014  }
1015
1016  }
1017
1017
1018  }
1019
1020  }
1021
1022  }
1023
1024  }
1025
1026  }
1027
1027
1028  }
1029
1030  }
1031
1032  }
1033
1034  }
1035
1036  }
1037
1037
1038  }
1039
1040  }
1041
1042  }
1043
1044  }
1045
1046  }
1047
1047
1048  }
1049
1050  }
1051
1052  }
1053
1054  }
1055
1056  }
1057
1057
1058  }
1059
1060  }
1061
1062  }
1063
1064  }
1065
1066  }
1067
1067
1068  }
1069
1070  }
1071
1072  }
1073
1074  }
1075
1076  }
1077
1077
1078  }
1079
1080  }
1081
1082  }
1083
1084  }
1085
1086  }
1087
1087
1088  }
1089
1090  }
1091
1092  }
1093
1094  }
1095
1096  }
1097
1097
1098  }
1099
1100  }
1101
1102  }
1103
1104  }
1105
1106  }
1107
1107
1108  }
1109
1110  }
1111
1112  }
1113
1114  }
1115
1116  }
1117
1117
1118  }
1119
1120  }
1121
1122  }
1123
1124  }
1125
1126  }
1127
1127
1128  }
1129
1130  }
1131
1132  }
1133
1134  }
1135
1136  }
1137
1137
1138  }
1139
1140  }
1141
1142  }
1143
1144  }
1145
1146  }
1147
1147
1148  }
1149
1150  }
1151
1152  }
1153
1154  }
1155
1156  }
1157
1157
1158  }
1159
1160  }
1161
1162  }
1163
1164  }
1165
1166  }
1167
1167
1168  }
1169
1170  }
1171
1172  }
1173
1174  }
1175
1176  }
1177
1177
1178  }
1179
1180  }
1181
1182  }
1183
1184  }
1185
1186  }
1187
1187
1188  }
1189
1190  }
1191
1192  }
1193
1194  }
1195
1196  }
1197
1197
1198  }
1199
1200  }
1201
1202  }
1203
1204  }
1205
1206  }
1207
1207
1208  }
1209
1210  }
1211
1212  }
1213
1214  }
1215
1216  }
1217
1217
1218  }
1219
1220  }
1221
1222  }
1223
1224  }
1225
1226  }
1227
1227
1228  }
1229
1230  }
1231
1232  }
1233
1234  }
1235
1236  }
1237
1237
1238  }
1239
1240  }
1241
1242  }
1243
1244  }
1245
1246  }
1247
1247
1248  }
1249
1250  }
1251
1252  }
1253
1254  }
1255
1256  }
1257
1257
1258  }
1259
1260  }
1261
1262  }
1263
1264  }
1265
1266  }
1267
1267
1268  }
1269
1270  }
1271
1272  }
1273
1274  }
1275
1276  }
1277
1277
1278  }
1279
1280  }
1281
1282  }
1283
1284  }
1285
1286  }
1287
1287
1288  }
1289
1290  }
1291
1292  }
1293
1294  }
1295
1296  }
1297
1297
1298  }
1299
1300  }
1301
1302  }
1303
1304  }
1305
1306  }
1307
1307
1308  }
1309
1310  }
1311
1312  }
1313
1314  }
1315
1316  }
1317
1317
1318  }
1319
1320  }
1321
1322  }
1323
1324  }
1325
1326  }
1327
1327
1328  }
1329
1330  }
1331
1332  }
1333
1334  }
1335
1336  }
1337
1337
1338  }
1339
1340  }
1341
1342  }
1343
1344  }
1345
1346  }
1347
1347
1348  }
1349
1350  }
1351
1352  }
1353
1354  }
1355
1356  }
1357
1357
1358  }
1359
1360  }
1361
1362  }
1363
1364  }
1365
1366  }
1367
1367
1368  }
1369
1370  }
1371
1372  }
1373
1374  }
1375
1376  }
1377
1377
1378  }
1379
1380  }
1381
1382  }
1383
1384  }
1385
1386  }
1387
1387
1388  }
1389
1390  }
1391
1392  }
1393
1394  }
1395
1396  }
1397
1397
1398  }
1399
1400  }
1401
1402  }
1403
1404  }
1405
1406  }
1407
1407
1408  }
1409
1410  }
1411
1412  }
1413
1414  }
1415
1416  }
1417
1417
1418  }
1419
1420  }
1421
1422  }
1423
1424  }
1425
1426  }
1427
1427
1428  }
1429
1430  }
1431
1432  }
1433
1434  }
1435
1436  }
1437
1437
1438  }
1439
1440  }
1441
1442  }
1443
1444  }
1445
1446  }
1447
1447
1448  }
1449
1450  }
1451
1452  }
1453
1454  }
1455
1456  }
1457
1457
1458  }
1459
1460  }
1461
1462  }
1463
1464  }
1465
1466  }
1467
1467
1468  }
1469
1470  }
1471
1472  }
1473
1474  }
1475
1476  }
1477
1477
1478  }
1479
1480  }
1481
1482  }
1483
1484  }
1485
1486  }
1487
1487
1488  }
1489
1490  }
1491
1492  }
1493
1494  }
1495
1496  }
1497
1497
1498  }
1499
1500  }
1501
1502  }
1503
1504  }
1505
1506  }
1507
1507
1508  }
1509
1510  }
1511
1512  }
1513
1514  }
1515
1516  }
1517
1517
1518  }
1519
1520  }
1521
1522  }
1523
1524  }
1525
1526  }
1527
1527
1528  }
1529
1530  }
1531
1532  }
1533
1534  }
1535
1536  }
1537
1537
1538  }
1539
1540  }
1541
1542  }
1543
1544  }
1545
1546  }
1547
1547
1548  }
1549
1550  }
1551
1552  }
1553
1554  }
1555
1556  }
1557
1557
1558  }
1559
1560  }
1561
1562  }
1563
1564  }
1565
1566  }
1567
1567
1568  }
1569
1570  }
1571
1572  }
1573
1574  }
1575
1576  }
1577
1577
1578  }
1579
1580  }
1581
1582  }
1583
1584  }
1585
1586  }
1587
1587
1588  }
1589
1590  }
1591
1592  }
1593
1594  }
1595
1596  }
1597
1597
1598  }
1599
1600  }
1601
1602  }
1603
1604  }
1605
1606  }
1607
1607
1608  }
1609
1610  }
1611
1612  }
1613
1614  }
1615
1616  }
1617
1617
1618  }
1619
1620  }
1621
1622  }
1623
1624  }
1625
1626  }
1627
1627
1628  }
1629
1630  }
1631
1632  }
1633
1634  }
1635
1636  }
1637
1637
1638  }
1639
1640  }
1641
1642  }
1643
1644  }
1645
1646  }
1647
1647
1648  }
1649
1650  }
1651
1652  }
1653
1654  }
1655
1656  }
1657
1657
1658  }
1659
1660  }
1661
1662  }
1663
1664  }
1665
1666  }
1667
1667
1668  }
1669
1670  }
1671
1672  }
1673
1674  }
1675
1676  }
1677
1677
1678  }
1679
1680  }
1681
1682  }
1683
1684  }
1685
1686  }
1687
1687
1688  }
1689
1690  }
1691
1692  }
1693
1694  }
1695
1696  }
1697
1697
1698  }
1699
1700  }
1701
1702  }
1703
1704  }
1705
1706  }
1707
1707
1708  }
1709
1710  }
1711
1712  }
1713
1714  }
1715
1716  }
1717
1717
1718  }
1719
1720  }
1721
1722  }
1723
1724  }
1725
1726  }
1727
1727
1728  }
1729
1730  }
1731
1732  }
1733
1734  }
1735
1736  }
1737
1737
1738  }
1739
1740  }
1741
1742  }
1743
1744  }
1745
1746  }
1747
1747
1748  }
1749
1750  }
1751
1752  }
1753
1754  }
1755
1756  }
1757
1757
1758  }
1759
1760  }
1761
1762  }
1763
1764  }
1765
1766  }
1767
1767
1768  }
1769
1770  }
1771
1772  }
1773
1774  }
1775
1776  }
1777
1777
1778  }
1779
1780  }
1781
1782  }
1783
1784  }
1785
1786  }
1787
1787
1788  }
1789
1790  }
1791
1792  }
1793
1794  }
1795
1796  }
1797
1797
1798  }
1799
1800  }
1801
1802  }
1803
1804  }
1805
1806  }
1807
1807
1808  }
1809
1810  }
1811
1812  }
1813
1814  }
1815
1816  }
1817
1817
1818  }
1819
1820  }
1821
1822  }
1823
1824  }
1825
1826  }
1827
1827
1828  }
1829
1830  }
1831
1832  }
1833
1834  }
1835
1836  }
1837
1837
1838  }
1839
1840  }
1841
1842  }
1843
1844  }
1845
1846  }
1847
1847
1848  }
1849
1850  }
1851
1852  }
1853
1854  }
1855
1856  }
1857
1857
1858  }
1859
1860  }
1861
1862  }
1863
1864  }
1865
1866  }
1867
1867
1868  }
1869
1870  }
1871
1872  }
1873
1874  }
1875
1876  }
1877
1877
1878  }
1879
1880  }
1881
1882  }
1883
1884  }
1885
1886  }
1887
1887
1888  }
1889
1890  }
1891
1892  }
1893
1894  }
1895
1896  }
1897
1897
1898  }
1899
1900  }
1901
1902  }
1903
1904  }
1905
1906  }
1907
1907
1908  }
1909
1910  }
1911
1912  }
1913
1914  }
1915
1916  }
1917
1917
1918  }
1919
1920  }
1921
1922  }
1923
1924  }
1925
1926  }
1927
1927
1928  }
1929
1930  }
1931
1932  }
1933
1934  }
1935
1936  }
1937
1937
1938  }
1939
1940  }
1941
1942  }
1943
1944  }
1945
1946  }
1947
1947
1948  }
1949
1950  }
1951
1952  }
1953
1954  }
1955
1956  }
1957
1957
1958  }
1959
1960  }
1961
1962  }
1963
1964  }
1965
1966  }
1967
1967
1968  }
1969
1970  }
1971
1972  }
1973
1974  }
1975
1976  }
1977
1977
1978  }
1979
1980  }
1981
1982  }
1983
1984  }
1985
1986  }
1987
1987
1988  }
1989
1990  }
1991
1992  }
1993
1994  }
1995
1996  }
1997
1997
1998  }
1999
2000  }
2001
2002  }
2003
2004  }
2005
2006  }
2007
2007
2008  }
2009
2010  }
2011
2012  }
2013
2014  }
2015
2016  }
2017
2017
2018  }
2019
2020  }
2021
2022  }
2023
2024  }
2025
2026  }
2027
2027
2028  }
2029
2030  }
2031
2032  }
2033
2034  }
2035
2036  }
2037
2037
2038  }
2039
2040  }
2041
2042  }
2043
2044  }
2045
2046  }
2047
2047
2048  }
2049
2050  }
2051
2052  }
2053
2054  }
2055
2056  }
2057
2057
2058  }
2059
2060  }
2061
2062  }
2063
2064  }
2065
2066  }
2067
2067
2068  }
2069
2070  }
2071
2072  }
2073
2074  }
2075
2076  }
2077
2077
2078  }
2079
2080  }
2081
2082  }
2083
2084  }
2085
2086  }
2087
2087
2088  }
2089
2090  }
2091
2092  }
2093
2094  }
2095
2096  }
2097
2097
2098  }
2099
2100  }
2101
2102  }
2103
2104  }
2105
2106  }
2107
2107
2108  }
2109
2110  }
2111
2112  }
2113
2114  }
2115
2116  }
2117
2117
2118  }
2119
2120  }
2121
2122  }
2123
2124  }
2125
2126  }
2127
2127
2128  }
2129
2130  }
2131
2132  }
2133
2134  }
2135
2136  }
2137
2137
2138  }
2139
2140  }
2141
2142  }
2143
2144  }
2145
2146  }
2147
2147
2148  }
2149
2150  }
2151
2152  }
2153
2154  }
2155
2156  }
2157
2157
2158  }
2159
2160  }
2161
2162  }
2163
2164  }
2165
2166  }
2167
2167
2168  }
2169
2170  }
2171
2172  }
2173
2174  }
2175
2176  }
2177
2177
2178  }
2179
2180  }
2181
2182  }
2183
2184  }
2185
2186  }
2187
2187
2188  }
2189
2190  }
2191
2192  }
2193
2194  }
2195
2196  }
2197
2197
2198  }
2199
2200  }
2201
2202  }
2203
2204  }
2205
2206  }
2207
2207
2208  }
2209
2210  }
2211
2212  }
2213
2214  }
2215
2216  }
2217
2217
2218  }
2219
2220  }
2221
2222  }
2223
2224  }
2225
2226  }
2227
2227
2228  }
2229
2230  }
2231
2232  }
2233
2234  }
2235
2236  }
2237
2237
2238  }
2239
2240  }
2241
2242  }
2243
2244  }
2245
2246  }
2247
2247
2248  }
2249
2250  }
2251
2252  }
2253
2254  }
2255
2256  }
2257
2257
2258  }
2259
2260  }
2261
2262  }
2263
2264  }
2265
2266  }
2267
2267
2268  }
2269
2270  }
2271
2272  }
2273
2274  }
2275
2276  }
2277
2277
2278  }
2279
2280  }
2281
2282  }
2283
2284  }
2285
2286  }
2287
2287
2288  }
2289
2290  }
2291
2292  }
2293
2294  }
2295
2296  }
2297
2297
2298  }
2299
2300  }
2301
2302  }
2303
2304  }
2305
2306  }
2307
2307
2308  }
2309
2310  }
2311
2312  }
2313
2314  }
2315
2316  }
2317
2317
2318  }
2319
2320  }
2321
2322  }
2323
2324  }
2325
2326  }
2327
2327
2328  }
2329
2330  }
2331
2332  }
2333
2334  }
2335
2336  }
2337
2337
2338  }
2339
2340  }
2341
2342  }
2343
2344  }
2345
2346  }
2347
2347
2348  }
2349
2350  }
2351
2352  }
2353
2354  }
2355
2356  }
2357
2357
2358  }
2359
2360  }
2361
2362  }
2363
2364  }
2365
2366  }
2367
2367
2368  }
2369
2370  }
2371
2372  }
2373
2374  }
2375
2376  }
2377
2377
2378  }
2379
2380  }
2381
2382  }
2383
2384  }
2385
2386  }
2387
2387
2388  }
2389
2390  }
2391
2392  }
2393
2394  }
2395
2396  }
2397
2397
2398  }
2399
2400  }
2401
2402  }
2403
2404  }
2405
2406  }
2407
2407
2408  }
2409
2410  }
2411
2412  }
2413
2414  }
2415
2416  }
2417
2417
2418  }
2419
2420  }
2421
2422  }
2423
2424  }
2425
2426  }
2427
2427
2428  }
2429
2430  }
2431
2432  }
2433
2434  }
2435
2436  }
2437
2437
24
```

552	const resolveDuration = (item: LineItemLike)	427	const resolveDuration = (item: LineItemLike): { value: number; fromData: boolean } => {
553	item: LineItemLike		
554): { value: number; fromData: boolean } => {		
555	const baseline = item.baseline as LineItemLike undefined;	428	const baseline = item.baseline as LineItemLike undefined;
556	const candidates = [429	const candidates = [
557	item.duration,	430	item.duration,
...	@@ -571,20 +444,20 @@ export function SDMTForecast() {		
571	};	444	};
572		445	
573	const resolveString = (value: unknown): string undefined => {	446	const resolveString = (value: unknown): string undefined => {
574	if (typeof value === "string" && value.trim().length > 0) {	447	if (typeof value === "string" && value.trim().length > 0) {
575	return value;	448	return value;
576	}	449	}
577	return undefined;	450	return undefined;
578	};	451	};
579		452	
580	lineItems.forEach((item) => {	453	lineItems.forEach(item => {
581	const lineItemId =	454	const lineItemId =
582	resolveString(item.id)	455	resolveString(item.id)
583	resolveString(item.rubroId)	456	resolveString(item.rubroId)
584	resolveString(item.rubro_id)	457	resolveString(item.rubro_id)
585	resolveString(item.linea_codigo)	458	resolveString(item.linea_codigo)
586	resolveString(item.linea_id)	459	resolveString(item.linea_id)
587	";	460	";
588		461	
589	if (!lineItemId) {	462	if (!lineItemId) {
590	return;	463	return;
...	@@ -599,33 +472,20 @@ export function SDMTForecast() {		
599	const duration = resolveDuration(item);	472	const duration = resolveDuration(item);
600	const startMonthRaw =	473	const startMonthRaw = resolveNumber(item.start_month, NaN);
601	resolveNumber(item.start_month, NaN);	474	const endMonthRaw = resolveNumber(item.end_month, NaN);
602	const endMonthRaw = resolveNumber(item.end_month,	475	const hasExplicitRange = Number.isFinite(startMonthRaw)
603	NaN);		Number.isFinite(endMonthRaw);
604	const startMonth = hasExplicitRange	476	const startMonth = hasExplicitRange

605	<pre>? Math.max(1, Math.min(months, Number.isFinite(startMonthRaw) ? Math.trunc(startMonthRaw) : 1))</pre>	477	<pre>? Math.max(1, Math.min(months, Number.isFinite(startMonthRaw) ? Math.trunc(startMonthRaw) : 1))</pre>
612	<pre>: 1;</pre>	478	<pre>: 1;</pre>
613	<pre>const endMonth = hasExplicitRange</pre>	479	<pre>const endMonth = hasExplicitRange</pre>
614	<pre>? Math.max(startMonth, Math.min(months, Number.isFinite(endMonthRaw) ? Math.trunc(endMonthRaw) : months))</pre>	480	<pre>? Math.max(startMonth, Math.min(months, Number.isFinite(endMonthRaw) ? Math.trunc(endMonthRaw) : months))</pre>
621	<pre>: duration.fromData</pre>	481	<pre>: duration.fromData</pre>
622	<pre>? Math.min(months, duration.value)</pre>	482	<pre>? Math.min(months, duration.value)</pre>
623	<pre>: months;</pre>	483	<pre>: months;</pre>
624		484	
625	<pre>const resolveFirstNumber = (values: unknown[]): number null => { for (const value of values) { if (value === null value === undefined) continue; if (typeof value === "string" && value.trim().length === 0) continue; const numeric = Number(value); if (Number.isFinite(numeric)) return numeric; } }</pre>	485	<pre>const resolveFirstNumber = (values: unknown[]): number null => { for (const value of values) { if (value === null value === undefined) continue; if (typeof value === "string" && value.trim().length === 0) continue; const numeric = Number(value); if (Number.isFinite(numeric)) return numeric; } }</pre>
...	<pre>@@ -645,8 +505,7 @@ export function SDMTForecast() {</pre>	505	<pre>item.unitCost ?? item.costo_unitario ?? item.unit_cost,</pre>
645	<pre>item.unit_cost, 0);</pre>	506	<pre>0</pre>
646		507	<pre>);</pre>
648	<pre>const qty = resolveNumber(item.qty ?? item.cantidad ?? item.quantity, 1)</pre>	508	<pre>const qty = resolveNumber(item.qty ?? item.cantidad ?? item.quantity, 1) 1;</pre>
649	<pre>item.quantity, 1) 1;</pre>		

650 const computedTotal = baseTotal ?? unitCost * qty;	509 const computedTotal = baseTotal ?? unitCost * qty;
651 const activeMonthCount = Math.max(1, endMonth -	510 const activeMonthCount = Math.max(1, endMonth - startMonth + 1);
652 startMonth + 1);	511 const defaultMonthlyAmount = computedTotal / activeMonthCount;
653 const defaultMonthlyAmount = computedTotal / activeMonthCount;	
654 ... @@ -661,33 +520,19 @@ export function SDMTForecast() {	
661 ? monthlyOverride	520 ? monthlyOverride
662 : defaultMonthlyAmount;	521 : defaultMonthlyAmount;
663	522
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	
671 taxonomy.descripcion : '')	
672 lineItemId;	
673 const category =	
674 resolveString(item.category)	
675 resolveString(item.categoría)	
676 (taxonomy ? taxonomy.categoría : ''));	
677	
678 forecastCells.push({	523 forecastCells.push({
679 line_item_id: lineItemId,	524 line_item_id: lineItemId,
680 rubroId: rubroKey,	525 rubroId: rubroKey,
681 projectId,	526 projectId,
682 description,	527 description: resolveString(item.description)
683 category,	528 resolveString(item.descripcion) '',
684 month,	529 category: resolveString(item.category)
685 planned: amount,	530 resolveString(item.categoría) '',
686 forecast: amount,	531 month,
687 actual: 0,	532 planned: amount,
688 variance: 0,	533 forecast: amount,
689 last_updated: new Date().toISOString(),	534 actual: 0,
690 updated_by: user?.email user?.login	535 variance: 0,
"system",	last_updated: new Date().toISOString(),


```
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,
585             aborting`);
586         }
587     }
588
589     // Extract results safely with strong type guards
590     const rubrosRes = results[0].status === 'fulfilled' &&
591     Array.isArray(results[0].value) ? results[0].value as any[] : [];
592     const allocationsRes = results[1].status === 'fulfilled' &&
593     Array.isArray(results[1].value) ? results[1].value as any[] : [];
594     const prefacturasRes = results[2].status === 'fulfilled' &&
595     Array.isArray(results[2].value) ? results[2].value as any[] : [];
596     const baselineRes: BaselineDetailResponse | null = (results[3] &&
597     results[3].status === 'fulfilled') ? results[3].value as BaselineDetailResponse
598     : null;
599
600     // Update baseline detail
601     setBaselineDetail(baselineRes);
602
603     if (import.meta.env.DEV) {
604         console.debug(`[loadBaselineRubros] Results:`, {
605             rubros: rubrosRes?.length || 0,
606             allocations: allocationsRes?.length || 0,
607             prefacturas: prefacturasRes?.length || 0,
608             baseline: baselineRes ? 'loaded' : 'null',
609         });
610     }
611
612     // If rubros endpoint returned data, use it
613     if (rubrosRes && rubrosRes.length > 0) {
614         setRubrosSource('api');
615         if (import.meta.env.DEV) {
616             console.debug(`[loadBaselineRubros] Using ${rubrosRes.length} rubros
from API`);
```

713

```
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 }
```

```

647     };
648
649     const loadSingleProjectForecast = async (projectId: string, months: number,
650     requestKey: string) => {
651         const payload = await getForecastPayload(projectId, months);
652
653         // Check if request is still valid before continuing
654         if (latestRequestKeyRef.current !== requestKey) {
655             return; // Stale, abort processing
656         }
657         // Pass baseline ID and enable debug mode in development for better
658         // diagnostics
659         const debugMode = import.meta.env.DEV;
660         let normalized = normalizeForecastCells(payload.data, {
661             baselineId: currentProject?.baselineId,
662             debugMode,
663         });
664         let usedFallback = false;
665         const baselineStatus = resolveBaselineStatus(
666             currentProject as { baselineStatus?: string; baseline_status?: string } | null
667         );
668         const hasAcceptedBaseline = baselineStatus === 'accepted';
669         // Fallback: If server forecast is empty and we have line items, use them
670         if (
671             (!normalized || normalized.length === 0) &&
672             ...
673             @-743,43 +678,38 @@ export function SDMTForecast() {
674                 `'[SDMTForecast] Using baseline fallback for
675                 ${projectId}, baseline ${currentProject?.baselineId}:
676                 ${safeLineItems.length} line items`:
677             );
678         );
679         normalized = transformLineItemsToForecast(safeLineItems, months,
680         projectId);

```

747	safeLineItems,	
748	months,	
749	projectId	
750);	
751	usedFallback = true;	682 usedFallback = true;
752	} else if (normalized && normalized.length > 0 && import.meta.env.DEV) {	683 } else if (normalized && normalized.length > 0 && import.meta.env.DEV) {
753	console.debug(`	684 console.debug(`
754	'[SDMTForecast] Using server forecast rows for \${projectId}, baseline \${currentProject?.baselineId}'`	685 '[SDMTForecast] Using server forecast rows for \${projectId}, baseline \${currentProject?.baselineId}'`
755);	686);
756	}	687 }
757		688
758	setDataSource(usedFallback ? "mock" : payload.source);	689 setDataSource(usedFallback ? 'mock' : payload.source); // Mark as 'mock' to indicate fallback
759	// Mark as 'mock' to indicate fallback	
760	setGeneratedAt(payload.generatedAt);	690 setGeneratedAt(payload.generatedAt);
761	setPortfolioLineItems([]);	691 setPortfolioLineItems([]);
762	// Get matched invoices and sync with actuals	692 // Get matched invoices and sync with actuals
763	const invoices = await getProjectInvoices(projectId);	693 const invoices = await getProjectInvoices(projectId);
764		694
765	// Check again after async operation	695 // Check again after async operation
766	if (latestRequestKeyRef.current !== requestKey) {	696 if (latestRequestKeyRef.current !== requestKey) {
767	return;	697 return;
768	}	698 }
769		699
770	const matchedInvoices = invoices.filter(inv =>	700 //
771	inv.status === "Matched");	701 const matchedInvoices = invoices.filter(inv => inv.status === 'Matched');
772	const updatedData: ForecastRow[] =	702
773	normalized.map((cell) => {	703 const updatedData: ForecastRow[] = normalized.map(cell => {
774	const matchedInvoice = matchedInvoices.find(704 const matchedInvoice = matchedInvoices.find(inv =>
775	(inv) =>	705 inv.line_item_id === cell.line_item_id && inv.month === cell.month
776	inv.line_item_id === cell.line_item_id &&	
777	inv.month === cell.month	
778);	706);
779	const withActuals = matchedInvoice	707
	? {	708 const withActuals = matchedInvoice
		709 ? {

780 ...cell,	710 ...cell,
781 actual: matchedInvoice.amount 0,	711 actual: matchedInvoice.amount 0,
782 variance: cell.forecast - cell.planned, // Keep forecast-based variance	712 variance: cell.forecast - cell.planned // Keep forecast-based
783 }	713 }
784 : cell;	714 : cell;
785	715
...	...
791 });	721 });
792	722
793 if (import.meta.env.DEV) {	723 if (import.meta.env.DEV) {
794 console.debug("[Forecast] data pipeline", {	724 console.debug("[Forecast] data pipeline", {
795 projectId,	725 projectId,
796 rawCells: Array.isArray(payload.data) ?	726 rawCells: Array.isArray(payload.data) ? payload.data.length : 0,
payload.data.length : 0,	normalizedCells: normalized.length,
797 normalizedCells: normalized.length,	727 normalizedCells: normalized.length,
...	...
811 setForecastData(updatedData);	741 setForecastData(updatedData);
812	742
813 if (import.meta.env.DEV) {	743 if (import.meta.env.DEV) {
814 console.debug("[Forecast] Data loaded", {	744 console.debug("[Forecast] Data loaded", {
815 projectId,	745 projectId,
816 source: usedFallback ? "lineItems-fallback" : payload.source,	746 source: usedFallback ? "lineItems-fallback" : payload.source,
817 records: updatedData.length,	747 records: updatedData.length,
818 });	748 });
819 }	749 }
820 };	750 };
821	751
822 const loadPortfolioForecast = async (months: number, requestKey: string) => {	752 const loadPortfolioForecast = async (months: number, requestKey: string) => {
823 if (import.meta.env.DEV) {	753 const candidateProjects = projects.filter(project => project.id &&
824 console.log(`[loadPortfolioForecast] projects loaded: \${projects.length}, ids: \${JSON.stringify(projects.map(p => p.id))}`)	project.id !== ALL_PROJECTS_ID);
825 `	
826);	
827 }	
828	
829 const candidateProjects = projects.filter(

830	(project) => project.id && project.id !== ALL_PROJECTS_ID);	
832	// TODO(SDMT): Replace per-project fan-out with aggregate portfolio endpoints when available.	754 // TODO(SDMT): Replace per-project fan-out with aggregate portfolio endpoints when available.
833		755
834	// Guard: Don't error out if projects haven't loaded yet	756 // 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) {	760 if (projects.length < MINIMUM_PROJECTS_FOR_PORTFOLIO) {
839	// Projects might still be loading; don't set error yet	761 // Projects might still be loading; don't set error yet
840	if (import.meta.env.DEV) {	762 if (import.meta.env.DEV) {
841	console.debug("[Forecast] Portfolio: Waiting for projects to load...");	763 console.debug('[Forecast] Portfolio: Waiting for projects to load...');
842		
843) ;	
844	}	764 }
845	setForecastData([]);	765 setForecastData([]);
846	return;	766 return;
847	}	767 }
848	// If we have projects but they're all filtered out, that's a real empty state	768 // If we have projects but they're all filtered out, that's a real empty state
849	setForecastError("No hay proyectos disponibles para consolidar.");	769 setForecastError('No hay proyectos disponibles para consolidar.');
850	setForecastData([]);	770 setForecastData([]);
851	return;	771 return;
852	}	772 }
853		773
854	const portfolioResults = await Promise.all(774 const portfolioResults = await Promise.all(
855	candidateProjects.map(async (project) => {	775 candidateProjects.map(async project => {
856	try {	776 try {
857	const [payload, invoices, projectLineItems] = await Promise.all([777 const [payload, invoices, projectLineItems] = await Promise.all([
858	getForecastPayload(project.id, months),	778 getForecastPayload(project.id, months),
859	getProjectInvoices(project.id),	779 getProjectInvoices(project.id),
860	getProjectRubros(project.id).catch(() => [] as LineItem[]),	780 getProjectRubros(project.id).catch(() => [] as LineItem[]),
861]);	781]);
862		782
863	// Check if request is still valid after async operations	783 // Check if request is still valid after async operations

864 if (latestRequestKeyRef.current !== requestKey)	784 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;	789 const debugMode = import.meta.env.DEV;
870 let normalized =	790 let normalized = normalizeForecastCells(payload.data, {
normalizeForecastCells(payload.data, {	baselineId: project.baselineId,
871 baselineId: project.baselineId,	791 baselineId: project.baselineId,
872 debugMode,	792 debugMode
873 });	793 });
874 const baselineStatus = resolveBaselineStatus(794 const baselineStatus = resolveBaselineStatus(
875 project as {	795 project as { baselineStatus?: string; baseline_status?: string }
876 baselineStatus?: string;	
877 baseline_status?: string;	
878 } null	
879);	796);
880 const hasAcceptedBaseline = baselineStatus ===	797 const hasAcceptedBaseline = baselineStatus === 'accepted';
"accepted";	
881 let usedFallback = false;	798 let usedFallback = false;
882	799
883 if (800 if (!normalized normalized.length === 0) &&
884 (!normalized normalized.length === 0) &&	projectLineItems.length > 0 && hasAcceptedBaseline) {
885 projectLineItems.length > 0 &&	
886 hasAcceptedBaseline	
887) {	
888 if (import.meta.env.DEV) {	801 if (import.meta.env.DEV) {
889 console.debug(802 console.debug(
890 `[SDMTForecast] Using baseline fallback	803 ` [SDMTForecast] Using baseline fallback for \${project.id},
for \${project.id}, baseline \${project.baselineId}:	baseline \${project.baselineId}: \${projectLineItems.length} line items`
\${projectLineItems.length} line items`	
891);	804);
892 }	805 }
893 normalized = transformLineItemsToForecast(806 normalized = transformLineItemsToForecast(projectLineItems, months,
894 projectLineItems,	project.id);
895 months,	
896 project.id	
897);	
898 usedFallback = true;	807 usedFallback = true;

```

899         } else if (normalized.length > 0 &&
900           import.meta.env.DEV) {
901             console.debug(
902               `[SDMTForecast] Using server forecast rows
903               for ${project.id}, baseline ${project.baselineId}`
904             );
905             const matchedInvoices = invoices.filter(
906               (inv) => inv.status === "Matched"
907             );
908             const projectData: ForecastRow[] =
909               normalized.map((cell) => {
910                 const matchedInvoice =
911                   matchedInvoices.find((inv) => {
912                     const invoiceMonth =
913                       normalizeInvoiceMonth(inv.month);
914                     return (
915                       inv.line_item_id === cell.line_item_id &&
916                         invoiceMonth === cell.month
917                     );
918                   });
919                   const withActuals = matchedInvoice
920                   ? {
921                     ...cell,
922                     actual: matchedInvoice.amount || 0,
923                     variance: cell.forecast - cell.planned,
924                   }
925                   : cell;
926                   return {
927                     ...
928                     ...
929                     ...
930                   });
931                   if (import.meta.env.DEV) {
932                     console.log(
933                       `[loadPortfolioForecast] project
934                         ${project.id} forecastRows=${projectData.length}
808         } else if (normalized.length > 0 && import.meta.env.DEV) {
809           console.debug(
810             `[SDMTForecast] Using server forecast rows for ${project.id},
811             baseline ${project.baselineId}`
812           );
813           const matchedInvoices = invoices.filter(inv => inv.status ===
814             'Matched');
815           const projectData: ForecastRow[] = normalized.map(cell => {
816             const matchedInvoice = matchedInvoices.find(inv =>
817               inv.line_item_id === cell.line_item_id && inv.month ===
818               cell.month
819             );
820             const withActuals = matchedInvoice
821             ? { ...cell, actual: matchedInvoice.amount || 0, variance:
822               cell.forecast - cell.planned }
823             : cell;
824             return {
825               ...
826               ...
827               ...
828             };
829           });
830

```

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 }	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 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 }

969	const aggregatedData = validResults.flatMap((result) => result.data);	855	const aggregatedData = validResults.flatMap(result => result.data);
970	const aggregatedLineItems = validResults.flatMap((result) => result.lineItems);	856	const aggregatedLineItems = validResults.flatMap(result => result.lineItems);
971		857	const firstGeneratedAt = validResults.find(result => result.generatedAt)??.generatedAt;
972			
973	const firstGeneratedAt = validResults.find((result) => result.generatedAt)??.generatedAt;	858	
974			
975			
976			
977	setDataSource('api');	859	setDataSource('api');
978	setGeneratedAt(firstGeneratedAt new Date().toISOString());	860	setGeneratedAt(firstGeneratedAt new Date().toISOString());
979	setPortfolioLineItems(aggregatedLineItems);	861	setPortfolioLineItems(aggregatedLineItems);
980	setForecastData(aggregatedData);	862	setForecastData(aggregatedData);
981		863	
982	if (import.meta.env.DEV) {	864	if (import.meta.env.DEV) {
983	console.debug('[Forecast] Portfolio data loaded', { projects: candidateProjects.length, records: aggregatedData.length, lineItems: aggregatedLineItems.length, });	865	console.debug('[Forecast] Portfolio data loaded', { projects: candidateProjects.length, records: aggregatedData.length, lineItems: aggregatedLineItems.length, });
984		866	
985		867	
986		868	
987		869	
988	}	870	}
989	}	871	
990		872	
991	// Consolidated data loading effect: handles initial load, route changes, and event-driven refreshes	873	const handleCellEdit = (line_item_id: string, month: number, type: 'forecast' 'actual') => {
992	// This ensures forecast loads on:	874	const cell = forecastData.find(c => c.line_item_id === line_item_id && c.month === month);
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) {		

```
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)
1026 // or main deps change
1026 triggerLoad();
1027
1028 // Check for URL refresh parameter
1029 const urlParams = new
1030 URLSearchParams(location.search);
1031 const refreshParam = urlParams.get("_refresh");
1032 if (refreshParam && selectedProjectId) {
1033   if (import.meta.env.DEV) {
1034     console.log("⌚ Forecast: Refreshing after
1035 reconciliation (URL param)");
1036   }
1037   // Already triggered above, but this logs the reason
1038 }
1039
1040 // Guard to prevent repeated visibility-based
1041 // refreshes
```

```
1039     let didRefreshOnVisibility = false;
1040
1041     // Visibility change: when tab becomes visible again,
1042     // reload once
1043     const onVisibility = () => {
1044       if (document.visibilityState === 'visible' &&
1045         selectedProjectId && !didRefreshOnVisibility) {
1046         didRefreshOnVisibility = true;
1047         if (import.meta.env.DEV) {
1048           console.log("⌚ Forecast: Refreshing on
1049             visibility change");
1050         }
1051         triggerLoad();
1052       } else if (document.visibilityState === 'hidden') {
1053         // Reset the flag when tab becomes hidden so next
1054         // visibility will trigger refresh
1055         didRefreshOnVisibility = false;
1056       }
1057     };
1058     window.addEventListener('visibilitychange',
1059       onVisibility);
1060
1061     // Cleanup: remove event listeners and abort
1062     // outstanding request when component unmounts
1063     return () => {
1064       window.removeEventListener('visibilitychange',
1065       onVisibility);
1066       // Abort any outstanding request when component
1067       // unmounts
1068       if (abortControllerRef.current) {
1069         abortControllerRef.current.abort();
1070       }
1071     };
1072   },
1073   [
1074     loadForecastData,
1075     location.key,
1076     location.search,
1077     selectedProjectId,
1078     selectedPeriod,
1079     projectChangeCount,
1080     currentProject?.baselineId,
1081   ]);
1082 ];
```

```

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
1081         === month
1082       );
1083       setEditingCell({ line_item_id, month, type });
1084       const currentValue = type === "forecast" ?
1085         cell?.forecast : cell?.actual;
1086       setEditValue(currentValue?.toString() || "0");
1087     };
1088
1089     const handleCellSave = () => {
1090       if (editingCell) {
1091         let pendingChange: ForecastRow | null = null;
1092         const updatedData = forecastData.map((cell) => {
1093           if (
1094             cell.line_item_id === editingCell.line_item_id
1095             && cell.month === editingCell.month
1096           ) {
1097             const newValue = parseFloat(editValue) || 0;
1098             const updates =
1099               editingCell.type === "forecast"
1100                 ? { forecast: newValue, variance: newValue -
1101                   cell.planned }
1102                 : { actual: newValue };
1103             const nextCell: ForecastRow = {
1104               ...cell,
1105               ...updates,
1106               last_updated: new Date().toISOString(),
1107               updated_by: user?.login || "current-user",
1108             };
1109             // Track the pending change
1110             pendingChange = {
1111               ...pendingChange,
1112               [cell.line_item_id]: {
1113                 month: cell.month,
1114                 type: cell.type,
1115                 value: newValue
1116               }
1117             };
1118           }
1119         );
1120         setPendingChange(pendingChange);
1121       }
1122     };
1123   
```

1109	if (editingCell.type === "actual") {	898	if (editingCell.type === 'actual') {
1110	pendingChange = nextCell;	899	pendingChange = nextCell;
1111	} else if (editingCell.type === "forecast") {	900	} else if (editingCell.type === 'forecast') {
1112	pendingChange = nextCell;	901	pendingChange = nextCell;
1113	}	902	}
1114		903	
1115	return nextCell;	904	return nextCell;
1116	}	905	}
1117	return cell;	906	return cell;
1118	});	907	});
1119	setForecastData(updatedData);	908	setForecastData(updatedData);
1120		909	
1121	if (pendingChange) {	910	if (pendingChange) {
1122	const changeKey = `\${pendingChange.projectId selectedProjectId}-\${pendingChange.line_item_id}` -`\${pendingChange.month}`;	911	const changeKey = `\${pendingChange.projectId selectedProjectId}-\${pendingChange.line_item_id}-\${pendingChange.month}`;
1123		912	if (editingCell.type === 'actual') {
1124		913	setDirtyActuals(prev => ({ ...prev, [changeKey]: pendingChange as ForecastRow }));
1125	if (editingCell.type === "actual") {	914	} else if (editingCell.type === 'forecast') {
1126	setDirtyActuals((prev) => {	915	setDirtyForecasts(prev => ({ ...prev, [changeKey]: pendingChange as ForecastRow }));
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	}	916	}
1136	}	917	}
1137		918	
1138	setEditingCell(null);	919	setEditingCell(null);
1139	toast.success(`	920	toast.success(`\${editingCell.type === 'forecast' ? 'Pronóstico' : 'Real'} actualizado correctamente`);
1140	`\${		
1141	editingCell.type === "forecast" ? "Pronóstico" : "Real"		
1142	`} actualizado correctamente`		
1143);		
1144	}	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
1151             guardar");
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 ||
1160                 selectedProjectId;
1161                 const matchedLineItem =
1162                 lineItemsForGrid.find((item) => {
1163                     const lineItemProjectId = (item as {
1164                     projectId?: string })
1165                     .projectId;
1166
1167                     return (
1168                         item.id === cell.line_item_id &&
1169                         (!lineItemProjectId || lineItemProjectId ===
1170                         projectId)
1171                     );
1172
1173                     if (!projectId) return null;
1174
1175                     const currency = (matchedLineItem?.currency ||
1176
1177                     "USD") as PayrollActualInput["currency"];
1178
1179                     return {
1180                         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
938         const matchedLineItem = lineItemsForGrid.find(item => {
939           const lineItemProjectId = (item as { projectId?: string }).projectId;
940           return item.id === cell.line_item_id && (!lineItemProjectId ||
941             lineItemProjectId === projectId);
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 =>
1192 Boolean(row));
1193 if (payload.length === 0) {
1194   toast.error("No pudimos construir los datos para
1195   guardar en nómina.");
1196   return;
1197 }
1198 await bulkUploadPayrollActuals(payload);
1199 toast.success("Valores reales enviados a Nómina
1200 (DynamoDB)");
1201 setDirtyActuals({});
1202 } catch (error) {
1203   console.error("✖ Error al guardar valores reales",
1204   error);
1205   const message = handleFinanzasApiError(error, {
1206     onAuthError: login,
1207     fallback: "No pudimos guardar los valores
1208     reales.",
1209   });
1210   setForecastError((prev) => prev || message);
1211 } finally {
1212   setSavingActuals(false);
1213 }
1214 ...
1215 @@ -1213,7 +980,7 @@ export function SDMTForecast() {
1216   const handlePersistForecasts = async () => {
1217     const entries: ForecastRow[] =
1218       Object.values(dirtyForecasts);
1219     const handlePersistForecasts = async () => {
1220       const entries: ForecastRow[] = Object.values(dirtyForecasts);
1221       const handlePersistForecasts = async () => {
1222         const entries: ForecastRow[] = Object.values(dirtyForecasts);
1223       }
1224     }
1225   }
1226 }
```

1215 if (entries.length === 0) {	982 if (entries.length === 0) {
1216 toast.info("No hay cambios de pronóstico para guardar");	983 toast.info("No hay cambios de pronóstico para guardar");
1217 return;	984 return;
1218 }	985 }
1219	986
... @@ -1226,46 +993,45 @@ export function SDMTForecast() {	
1226 try {	993 try {
1227 // Group by project for API calls	994 // Group by project for API calls
1228 const byProject = new Map<string, ForecastRow[]>();	995 const byProject = new Map<string, ForecastRow[]>();
1229 entries.forEach((cell) => {	996 entries.forEach(cell => {
1230 const projectId = cell.projectId selectedProjectId;	997 const projectId = cell.projectId selectedProjectId;
1231 if (!projectId) return;	998 if (!projectId) return;
1232	999
1233 if (!byProject.has(projectId)) {	1000 if (!byProject.has(projectId)) {
1234 byProject.set(projectId, []);	1001 byProject.set(projectId, []);
1235 }	1002 }
1236 byProject.get(projectId)!.push(cell);	1003 byProject.get(projectId)!.push(cell);
1237 });	1004 });
1238	1005
1239 // Send updates per project using bulkUpsertForecast with monthIndex format	1006 // Send updates per project using bulkUpsertForecast with monthIndex format
1240 // The API expects: {items: [{rubroId, month: number, forecast}]}	1007 // The API expects: {items: [{rubroId, month: number (1-12), forecast}]}
1241 // We send monthIndex as a number, and the backend will compute the calendar month	1008 // We send monthIndex as a number, and the backend will compute the calendar month
1242 for (const [projectId, projectCells] of byProject.entries()) {	1009 for (const [projectId, projectCells] of byProject.entries()) {
1243 const items = projectCells.map((cell) => {	1010 const items = projectCells.map(cell => {
1244 // Validate month is in valid range (up to baseline duration, fallback 60)	1011 // Validate month is in valid range (1-12)
1245 const maxMonths = getBaselineDuration(baselineDetail);	1012 const monthIndex = Math.max(1, Math.min(12, cell.month));
1246 const monthIndex = Math.max(1, Math.min(maxMonths, cell.month));	
1247 return {	1013 return {
1248 rubroId: cell.line_item_id,	1014 rubroId: cell.line_item_id,
1249 month: monthIndex,	1015 month: monthIndex, // Send as numeric monthIndex (1-12)
1250 forecast: Number(cell.forecast) 0,	1016 forecast: Number(cell.forecast) 0,
1251 };	1017 };
1252 });	1018 });
1253	1019

```

1254         await finanzasClient.bulkUpsertForecast(projectId,
1255         items);
1256
1257         toast.success("Pronósticos ajustados guardados
1258         exitosamente");
1259
1260         // Reload forecast data to show persisted values
1261         await loadForecastData();
1262     } catch (error) {
1263         console.error("🔴 Error al guardar pronósticos",
1264         error);
1265         const message = handleFinanzasApiError(error, {
1266             onAuthError: login,
1267             fallback: "No pudimos guardar los pronósticos
1268             ajustados.",
1269         });
1270         setForecastError((prev) => prev || message);
1271     }
1272
1273     ...
1274     @@ -1289,17 +1055,15 @@ export function SDMTForecast() {
1275         setBudgetMissingYear(resolution.state.missingYear);
1276
1277         // If 404, it means no budget is set for this year –
1278         // that's okay
1279         if (resolution.status === "missing") {
1280             console.warn(
1281                 `[SDMTForecast] ⚠️ No annual budget configured
1282                 for ${year}`);
1283
1284             return;
1285         }
1286
1287         console.error("Error loading annual budget:",
1288         error);
1289         const message = handleFinanzasApiError(error, {
1290             onAuthError: login,
1291             fallback: "No pudimos cargar el presupuesto
1292             anual.",
1293         });
1294
1295         ...
1296
1297         await finanzasClient.bulkUpsertForecast(projectId,
1298         items);
1299
1300         toast.success("Pronósticos ajustados guardados
1301         exitosamente");
1302
1303         setDirtyForecasts({});
```

1304 toast.error(message);	1068 toast.error(message);
1305 } finally {	1069 } finally {
... @@ -1310,7 +1074,7 @@ export function SDMTForecast() {	
1310 // Load Budget Overview for KPIs (only in portfolio view)	1074 // Load Budget Overview for KPIs (only in portfolio view)
1311 const loadBudgetOverview = async (year: number) => {	1075 const loadBudgetOverview = async (year: number) => {
1312 if (!isPortfolioView) return;	1076 if (!isPortfolioView) return;
1313 try {	1077 try {
1314 const overview = await	1078 const overview = await
finanzasClient.getAllInBudgetOverview(year);	finanzasClient.getAllInBudgetOverview(year);
1316 if (!overview) {	1080 if (!overview) {
... @@ -1321,15 +1085,15 @@ export function SDMTForecast() {	
1321 return;	1085 return;
1322 }	1086 }
1323 setBudgetOverview(overview);	1087 setBudgetOverview(overview);
1324 console.log(`[SDMTForecast] Budget overview loaded:\${overview}`);	1088 console.log(`[SDMTForecast] Budget overview loaded:\${overview}`);
1325 } catch (error: any) {	1089 } catch (error: any) {
1326 if (isBudgetNotFoundError(error)) {	1090 if (isBudgetNotFoundError(error)) {
1327 console.warn(`[SDMTForecast] ⚠️ Budget overview not found for \${year}`);	1091 console.warn(`[SDMTForecast] ⚠️ Budget overview not found for \${year}`);
1328 setBudgetOverview(null);	1092 setBudgetOverview(null);
1329 return;	1093 return;
1330 }	1094 }
1331 // Don't show error to user, just log it – this is optional enhancement	1095 // Don't show error to user, just log it – this is optional enhancement
1332 console.error("Error loading budget overview:", error);	1096 console.error(`Error loading budget overview: \${error}`);
1333 setBudgetOverview(null);	1097 setBudgetOverview(null);
1334 }	1098 }
1335 };	1099 };
... @@ -1338,28 +1102,24 @@ export function SDMTForecast() {	
1338 const handleSaveAnnualBudget = async () => {	1102 const handleSaveAnnualBudget = async () => {
1339 const amount = parseFloat(budgetAmount);	1103 const amount = parseFloat(budgetAmount);
1340 if (isNaN(amount) amount < 0) {	1104 if (isNaN(amount) amount < 0) {
1341 toast.error("Por favor ingrese un monto válido");	1105 toast.error(`Por favor ingrese un monto válido`);
1342 return;	1106 return;
1343 }	1107 }
1344 setSavingBudget(true);	1108 setSavingBudget(true);
	1109

1346	<code>try {</code>	1110	<code>try {</code>
1347	<code> const result = await finanzasClient.putAllInBudget(</code>	1111	<code> const result = await finanzasClient.putAllInBudget(budgetYear, amount,</code>
1348	<code> budgetYear,</code>		<code> budgetCurrency);</code>
1349	<code> amount,</code>		
1350	<code> budgetCurrency</code>		
1351	<code>);</code>		
1352	<code> setBudgetLastUpdated(result.updated_at);</code>	1112	<code> setBudgetLastUpdated(result.updated_at);</code>
1353	<code> toast.success("Presupuesto anual guardado exitosamente");</code>	1113	<code> toast.success('Presupuesto anual guardado exitosamente');</code>
1354		1114	
1355	<code>// Reload budget and budget overview to update KPIs</code>	1115	<code>// Reload budget and budget overview to update KPIs</code>
1356	<code>await loadAnnualBudget(budgetYear);</code>	1116	<code>await loadAnnualBudget(budgetYear);</code>
1357	<code>await loadBudgetOverview(budgetYear);</code>	1117	<code>await loadBudgetOverview(budgetYear);</code>
1358	<code>} catch (error) {</code>	1118	<code>} catch (error) {</code>
1359	<code> console.error("Error saving annual budget:", error);</code>	1119	<code> console.error('Error saving annual budget:', error);</code>
1360	<code> const message = handleFinanzasApiError(error, {</code>	1120	<code> const message = handleFinanzasApiError(error, {</code>
1361	<code> onAuthError: login,</code>	1121	<code> onAuthError: login,</code>
1362	<code> fallback: "No pudimos guardar el presupuesto anual.",</code>	1122	<code> fallback: 'No pudimos guardar el presupuesto anual.',</code>
1363	<code> });</code>	1123	<code>});</code>
1364	<code> toast.error(message);</code>	1124	<code> toast.error(message);</code>
1365	<code>} finally {</code>	1125	<code>} finally {</code>
...	<code>@@ -1370,7 +1130,7 @@ export function SDMTForecast() {</code>		
1370	<code>// Load Monthly Budget</code>	1130	<code>// Load Monthly Budget</code>
1371	<code>const loadMonthlyBudget = async (year: number) => {</code>	1131	<code>const loadMonthlyBudget = async (year: number) => {</code>
1372	<code> if (!isPortfolioView) return; // Monthly budgets only in portfolio view</code>	1132	<code> if (!isPortfolioView) return; // Monthly budgets only in portfolio view</code>
1373		1133	
1374	<code> setLoadingMonthlyBudget(true);</code>	1134	<code> setLoadingMonthlyBudget(true);</code>
1375	<code> try {</code>	1135	<code> try {</code>
1376	<code> const monthlyBudget = await finanzasClient.getAllInBudgetMonthly(year);</code>	1136	<code> const monthlyBudget = await finanzasClient.getAllInBudgetMonthly(year);</code>
...	<code>@@ -1383,21 +1143,19 @@ export function SDMTForecast() {</code>		
1383	<code> }</code>	1143	<code> }</code>
1384	<code> if (monthlyBudget && monthlyBudget.months) {</code>	1144	<code> if (monthlyBudget && monthlyBudget.months) {</code>
1385	<code> // Convert from API format (month: "YYYY-MM", amount) to internal format (month: 1-12, budget)</code>	1145	<code> // Convert from API format (month: "YYYY-MM", amount) to internal format (month: 1-12, budget)</code>
1386	<code> const budgets: MonthlyBudgetInput[] = monthlyBudget.months</code>	1146	<code> const budgets: MonthlyBudgetInput[] = monthlyBudget.months.map(m => {</code>
1387	<code> .map((m) => {</code>	1147	<code> const monthMatch = m.month.match(/^\d{4}-(\d{2})\$/);</code>

1388	const monthMatch = m.month.match(/^\d{4}- (\d{2})\$/); const monthNum = monthMatch ? parseInt(monthMatch[1], 10) : 0;	1148	const monthNum = monthMatch ? parseInt(monthMatch[1], 10) : 0;
1389		1149	return {
1390	return {	1150	month: monthNum,
1391	month: monthNum,	1151	budget: m.amount,
1392	budget: m.amount,	1152	};
1393	};	1153	}).filter(b => b.month >= 1 && b.month <= 12);
1394	}	1154	
1395	.filter((b) => b.month >= 1 && b.month <= 60); // Support up to 60 months		
1396			
1397	setMonthlyBudgets(budgets);	1155	setMonthlyBudgets(budgets);
1398	setMonthlyBudgetLastUpdated(monthlyBudget.updated_at null);	1156	setMonthlyBudgetLastUpdated(monthlyBudget.updated_at null);
1399	setMonthlyBudgetUpdatedBy(monthlyBudget.updated_by null);	1157	setMonthlyBudgetUpdatedBy(monthlyBudget.updated_by null);
1400		1158	
1401	// If we have saved monthly budgets, enable the monthly budget mode	1159	// If we have saved monthly budgets, enable the monthly budget mode
1402	if (budgets.length > 0) {	1160	if (budgets.length > 0) {
1403	setUseMonthlyBudget(true);	1161	setUseMonthlyBudget(true);
...	@@ -1421,13 +1179,11 @@ export function SDMTForecast() {		
1421	setMonthlyBudgetUpdatedBy(null);	1179	setMonthlyBudgetUpdatedBy(null);
1422	setUseMonthlyBudget(false);	1180	setUseMonthlyBudget(false);
1423	} else {	1181	} else {
1424	console.error("Error loading monthly budget:", error);	1182	console.error("Error loading monthly budget:", error);
1425		1183	
1426	// Show user-friendly error for network failures	1184	// Show user-friendly error for network failures
1427	if (error instanceof TypeError && error.message.includes("fetch")) {	1185	if (error instanceof TypeError && error.message.includes('fetch')) {
1428	toast.error("Error de red al cargar presupuesto mensual. Verifique la conexión e intente nuevamente."	1186	toast.error('Error de red al cargar presupuesto mensual. Verifique la conexión e intente nuevamente.');
1429)		
1430			
1431	}	1187	}
1432	// Don't show toast for other errors (optional feature, may not be configured)	1188	// Don't show toast for other errors (optional feature, may not be configured)
1433	}	1189	}
...	@@ -1436,60 +1192,6 @@ export function SDMTForecast() {	1192	}
1436	}		

```
1437      };
1438
1439      // Save Monthly Budget
1440      const handleSaveMonthlyBudget = async () => {
1441          if (monthlyBudgets.length === 0) {
1442              toast.error("Ingrese al menos un presupuesto
1443              mensual");
1444              return;
1445          }
1446
1447          setSavingMonthlyBudget(true);
1448          try {
1449              // Convert from internal format (month: 1-12,
1450              // budget) to API format (month: "YYYY-MM", amount)
1451              const months = monthlyBudgets.map((mb) => ({
1452                  month:
1453                      `${budgetYear}-${String(mb.month).padStart(2, "0")}`,
1454                  amount: mb.budget,
1455              }));
1456
1457              const result = await
1458                  finanzasClient.putAllInBudgetMonthly(
1459                      budgetYear,
1460                      budgetCurrency,
1461                      months
1462                  );
1463              setMonthlyBudgetLastUpdated(result.updated_at);
1464              setMonthlyBudgetUpdatedBy(result.updated_by);
1465              toast.success("Presupuesto mensual guardado
1466              exitosamente");
1467
1468              // Reload monthly budget and budget overview to
1469              // update KPIs and grid
1470              await loadMonthlyBudget(budgetYear);
1471              await loadBudgetOverview(budgetYear);
1472          } catch (error) {
1473              console.error("Error saving monthly budget:",
1474              error);
1475
1476              // Provide detailed error message for network
1477              // failures
1478              let message: string;
```

```
1193      };
1194
```

```
1471     if (error instanceof TypeError &&
1472     error.message.includes("fetch")) {
1473         message =
1474             "Error de red al guardar presupuesto mensual.
1475             Verifique la conexión, configuración de CORS, y la URL
1476             base de la API en las variables de entorno.";
1477         } else {
1478             message = handleFinanzasApiError(error, {
1479                 onAuthError: login,
1480                 fallback: "No pudimos guardar el presupuesto
1481                 mensual.",
1482             });
1483             toast.error(message);
1484         }
1485     } finally {
1486         setSavingMonthlyBudget(false);
1487     }
1488 }
1489
1490 // Reset monthly budget to auto-distribution
1491 const handleResetMonthlyBudget = () => {
1492     setMonthlyBudgets([]);
1493     setUseMonthlyBudget(false);
1494     toast.info("Presupuesto mensual restablecido a
1495     distribución automática");
1496 };
1497
1498 // Save budget from rubros table inline editing
1499 const handleSaveBudgetFromTable = async (
1500     budgets: Array<{ month: number; budget: number }>
...
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2298
2299
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2388
2389
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2398
2399
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2498
2499
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2529
2530
2531
2532
2533
2534
2535
2536
2537
2538
2539
2539
2540
2541
2542
2543
2544
2545
2546
2547
2548
2549
2549
2550
2551
2552
2553
2554
2555
2556
2557
2558
2559
2559
2560
2561
2562
2563
2564
2565
2566
2567
2568
2569
2569
2570
2571
2572
2573
2574
2575
2576
2577
2578
2579
2579
2580
2581
2582
2583
2584
2585
2586
2587
2588
2589
2589
2590
2591
2592
2593
2594
2595
2596
2597
2598
2598
2599
2599
2600
2601
2602
2603
2604
2605
2606
2607
2608
2609
2609
2610
2611
2612
2613
2614
2615
2616
2617
2618
2619
2619
2620
2621
2622
2623
2624
2625
2626
2627
2628
2629
2629
2630
2631
2632
2633
2634
2635
2636
2637
2638
2639
2639
2640
2641
2642
2643
2644
2645
2646
2647
2648
2649
2649
2650
2651
2652
2653
2654
2655
2656
2657
2658
2659
2659
2660
2661
2662
2663
2664
2665
2666
2667
2668
2669
2669
2670
2671
2672
2673
2674
2675
2676
2677
2678
2679
2679
2680
2681
2682
2683
2684
2685
2686
2687
2688
2689
2689
2690
2691
2692
2693
2694
2695
2696
2697
2698
2698
2699
2699
2700
2701
2702
2703
2704
2705
2706
2707
2708
2709
2709
2710
2711
2712
2713
2714
2715
2716
2717
2718
2719
2719
2720
2721
2722
2723
2724
2725
2726
2727
2728
2729
2729
2730
2731
2732
2733
2734
2735
2736
2737
2738
2739
2739
2740
2741
2742
2743
2744
2745
2746
2747
2748
2749
2749
2750
2751
2752
2753
2754
2755
2756
2757
2758
2759
2759
2760
2761
2762
2763
2764
2765
2766
2767
2768
2769
2769
2770
2771
2772
2773
2774
2775
2776
2777
2778
2779
2779
2780
2781
2782
2783
2784
2785
2786
2787
2788
2788
2789
2789
2790
2791
2792
2793
2794
2795
2796
2797
2797
2798
2799
2799
2800
2801
2802
2803
2804
2805
2806
2807
2808
2809
2809
2810
2811
2812
2813
2814
2815
2816
2817
2818
2819
2819
2820
2821
2822
2823
2824
2825
2826
2827
2828
2829
2829
2830
2831
2832
2833
2834
2835
2836
2837
2838
2839
2839
2840
2841
2842
2843
2844
2845
2846
2847
2848
2849
2849
2850
2851
2852
2853
2854
2855
2856
2857
2858
2859
2859
2860
2861
2862
2863
2864
2865
2866
2867
2868
2869
2869
2870
2871
2872
2873
2874
2875
2876
2877
2878
2879
2879
2880
2881
2882
2883
2884
2885
2886
2887
2888
2888
2889
2889
2890
2891
2892
2893
2894
2895
2896
2897
2897
2898
2899
2899
2900
2901
2902
2903
2904
2905
2906
2907
2908
2909
2909
2910
2911
2912
2913
2914
2915
2916
2917
2918
2919
2919
2920
2921
2922
2923
2924
2925
2926
2927
2928
2929
2929
2930
2931
2932
2933
2934
2935
2936
2937
2938
2939
2939
2940
2941
2942
2943
2944
2945
2946
2947
2948
2949
2949
2950
2951
2952
2953
2954
2955
2956
2957
2958
2959
2959
2960
2961
2962
2963
2964
2965
2966
2967
2968
2969
2969
2970
2971
2972
2973
2974
2975
2976
2977
2978
2979
2979
2980
2981
2982
2983
2984
2985
2986
2987
2988
2989
2989
2990
2991
2992
2993
2994
2995
2996
2997
2998
2998
2999
2999
3000
3001
3002
3003
3004
3005
3006
3007
3008
3009
3009
3010
3011
3012
3013
3014
3015
3016
3017
3018
3019
3019
3020
3021
3022
3023
3024
3025
3026
3027
3028
3029
3029
3030
3031
3032
3033
3034
3035
3036
3037
3038
3039
3039
3040
3041
3042
3043
3044
3045
3046
3047
3048
3049
3049
3050
3051
3052
3053
3054
3055
3056
3057
3058
3059
3059
3060
3061
3062
3063
3064
3065
3066
3067
3068
3069
3069
3070
3071
3072
3073
3074
3075
3076
3077
3078
3079
3079
3080
3081
3082
3083
3084
3085
3086
3087
3088
3088
3089
3089
3090
3091
3092
3093
3094
3095
3096
3097
3097
3098
3099
3099
3100
3101
3102
3103
3104
3105
3106
3107
3108
3109
3109
3110
3111
3112
3113
3114
3115
3116
3117
3118
3119
3119
3120
3121
3122
3123
3124
3125
3126
3127
3128
3129
3129
3130
3131
3132
3133
3134
3135
3136
3137
3138
3139
3139
3140
3141
3142
3143
3144
3145
3146
3147
3148
3149
3149
3150
3151
3152
3153
3154
3155
3156
3157
3158
3159
3159
3160
3161
3162
3163
3164
3165
3166
3167
3168
3169
3169
3170
3171
3172
3173
3174
3175
3176
3177
3178
3179
3179
3180
3181
3182
3183
3184
3185
3186
3187
3188
3189
3189
3190
3191
3192
3193
3194
3195
3196
3197
3197
3198
3199
3199
3200
3201
3202
3203
3204
3205
3206
3207
3208
3209
3209
3210
3211
3212
3213
3214
3215
3216
3217
3218
3219
3219
3220
3221
3222
3223
3224
3225
3226
3227
3228
3229
3229
3230
3231
3232
3233
3234
3235
3236
3237
3238
3239
3239
3240
3241
3242
3243
3244
3245
3246
3247
3248
3249
3249
3250
3251
3252
3253
3254
3255
3256
3257
3258
3259
3259
3260
3261
3262
3263
3264
3265
3266
3267
3268
3269
3269
3270
3271
3272
3273
3274
3275
3276
3277
3278
3279
3279
3280
3281
3282
3283
3284
3285
3286
3287
3288
3288
3289
3289
3290
3291
3292
3293
3294
3295
3296
3297
3297
3298
3299
3299
3300
3301
3302
3303
3304
3305
3306
3307
3308
3309
3309
3310
3311
3312
3313
3314
3315
3316
3317
3318
3319
3319
3320
3321
3322
3323
3324
3325
3326
3327
3328
3329
3329
3330
3331
3332
3333
3334
3335
3336
3337
3338
3339
3339
3340
3341
3342
3343
3344
3345
3346
3347
3348
3349
3349
3350
3351
3352
3353
3354
3355
3356
3357
3358
3359
3359
3360
3361
3362
3363
3364
3365
3366
3367
3368
3369
3369
3370
3371
3372
3373
3374
3375
3376
3377
3378
3379
3379
3380
3381
3382
3383
3384
3385
3386
3387
3388
3388
3389
3389
3390
3391
3392
3393
3394
3395
3396
3397
3397
3398
3399
3399
3400
3401
3402
3403
3404
3405
3406
3407
3408
3409
3409
3410
3411
3412
3413
3414
3415
3416
3417
3418
3419
3419
3420
3421
3422
3423
3424
3425
3426
3427
3428
3429
3429
3430
3431
3432
3433
3434
3435
3436
3437
3438
3439
3439
3440
3441
3442
3443
3444
3445
3446
3447
3448
3449
3449
3450
3451
3452
3453
3454
3455
3456
3457
3458
3459
3459
3460
3461
3462
3463
3464
3465
3466
3467
3468
3469
3469
3470
3471
3472
3473
3474
3475
3476
3477
3478
3479
3479
3480
3481
3482
3483
3484
3485
3486
3487
3488
3488
3489
3489
3490
3491
3492
3493
3494
3495
3496
3497
3497
3498
3499
3499
3500
3501
3502
3503
3504
3505
3506
3507
3508
3509
3509
3510
3511
3512
3513
3514
3515
3516
3517
3518
3519
3519
3520
3521
3522
3523
3524
3525
3526
3527
3528
3529
3529
3530
3531
3532
3533
3534
3535
3536
3537
3538
3539
3539
3540
3541
3542
3543
3544
3545
3546
3547
3548
3549
```