# Coordination UI Enhancements

## Notification Panel Grouping and Icons

Replace the **NotificationsBell.jsx** component with the following to group notifications by type and show an icon and timestamp for each:

import { useEffect, useMemo, useRef, useState } from "react";  
import { useQuery } from "@tanstack/react-query";  
import { Link } from "react-router-dom";  
import { apiService } from "../services/api";  
import { useToaster } from "./Toaster";  
  
export default function NotificationsBell() {  
 const [open, setOpen] = useState(false);  
 const lastSeen = useRef(0);  
 const { push } = useToaster();  
  
 const { data: items = [], refetch } = useQuery({  
 queryKey: ["recent\_notifications"],  
 queryFn: () => apiService.listRecentNotifications(),  
 refetchInterval: 30\_000,  
 });  
  
 useEffect(() => {  
 if (!items.length) return;  
 const latestTs = Math.max(...items.map(n => +new Date(n.created\_at)));  
 if (lastSeen.current && latestTs > lastSeen.current) {  
 push("success", "New notifications received");  
 }  
 lastSeen.current = latestTs;  
 }, [items, push]);  
  
 const unreadCount = useMemo(() => items.filter(n => !n.read\_at).length, [items]);  
  
 const handleMarkAsRead = async (id) => {  
 try {  
 await apiService.markNotificationRead(id);  
 push("success", "Notification marked as read");  
 refetch();  
 } catch {  
 push("error", "Failed to mark notification as read");  
 }  
 };  
  
 const handleMarkAllAsRead = async () => {  
 try {  
 await apiService.markAllNotificationsRead();  
 push("success", "All notifications marked as read");  
 refetch();  
 } catch {  
 push("error", "Failed to mark all as read");  
 }  
 };  
  
 // Group notifications by type  
 const grouped = useMemo(() => {  
 const groups = {};  
 for (let n of items) {  
 const t = n.type || "other";  
 if (!groups[t]) groups[t] = [];  
 groups[t].push(n);  
 }  
 return groups;  
 }, [items]);  
  
 const typeOrder = ["ball\_handoff", "comment\_added", "status\_changed"];  
 const groupEntries = Object.entries(grouped).sort(([a], [b]) => {  
 const ai = typeOrder.indexOf(a), bi = typeOrder.indexOf(b);  
 if (ai === -1 && bi === -1) return a.localeCompare(b);  
 if (ai === -1) return 1;  
 if (bi === -1) return -1;  
 return ai - bi;  
 });  
  
 const TYPE\_ICON = {  
 ball\_handoff: "🏀",  
 comment\_added: "💬",  
 status\_changed: "📊",  
 task\_created: "✨",  
 task\_assigned: "👤",  
 task\_overdue: "⏰",  
 };  
 const TYPE\_LABEL = {  
 ball\_handoff: "Ball Handoff",  
 comment\_added: "Comment",  
 status\_changed: "Status Change",  
 task\_created: "Task Created",  
 task\_assigned: "Task Assigned",  
 task\_overdue: "Task Overdue",  
 };  
  
 return (  
 <div className="relative">  
 <button   
 className="relative px-3 py-1 border rounded text-sm"   
 onClick={() => setOpen(o => !o)}  
 >  
 🔔 Notifications  
 {unreadCount > 0 && (  
 <span className="ml-2 inline-block text-xs bg-amber-500 text-white rounded-full px-2">  
 {unreadCount}  
 </span>  
 )}  
 </button>  
  
 {open && (  
 <div className="absolute right-0 mt-2 w-96 max-h-96 overflow-auto bg-white border rounded shadow z-40">  
 {/\* Panel Header \*/}  
 <div className="flex items-center justify-between px-3 py-2 border-b">  
 <span className="font-semibold text-sm">Recent Notifications</span>  
 <div className="flex gap-2">  
 {unreadCount > 0 && (  
 <button className="text-xs underline text-blue-600 hover:text-blue-800" onClick={handleMarkAllAsRead}>  
 Mark all read  
 </button>  
 )}  
 <button className="text-xs underline" onClick={() => refetch()}>Refresh</button>  
 </div>  
 </div>  
  
 {/\* Notification Groups \*/}  
 {items.length === 0 ? (  
 <div className="p-3 text-sm text-gray-500">No notifications.</div>  
 ) : (  
 groupEntries.map(([type, notifs]) => (  
 <div key={type}>  
 <div className="px-3 py-2 font-medium bg-gray-100 flex items-center gap-2">  
 <span>{TYPE\_ICON[type] || "🔔"}</span>  
 <span>{TYPE\_LABEL[type] || type}</span>  
 </div>  
 <ul className="divide-y">  
 {notifs.map(n => {  
 const isUnread = !n.read\_at;  
 return (  
 <li   
 key={n.id}   
 className={`p-3 text-sm flex items-start justify-between gap-3 ${isUnread ? "bg-blue-50" : "bg-white"}`}  
 >  
 <div className="flex-1">  
 <div className={isUnread ? "font-bold" : "font-medium"}>  
 {formatNotificationText(n)}  
 </div>  
 <div className="text-xs text-gray-600 mt-1">  
 {new Date(n.created\_at).toLocaleString()}  
 </div>  
 </div>  
 <div className="flex flex-col gap-1">  
 {n.task\_id && (  
 <Link   
 to={`/task/${n.task\_id}`}   
 className="px-2 py-1 border rounded text-xs bg-white hover:bg-gray-50 text-center"  
 onClick={() => setOpen(false)}  
 >  
 Open  
 </Link>  
 )}  
 {isUnread && (  
 <button   
 className="px-2 py-1 border rounded text-xs bg-white hover:bg-gray-50"  
 onClick={() => handleMarkAsRead(n.id)}  
 >  
 ✓ Read  
 </button>  
 )}  
 </div>  
 </li>  
 );  
 })}  
 </ul>  
 </div>  
 ))  
 )}  
 </div>  
 )}  
 </div>  
 );  
}  
  
// Helper to format notification text (preserves emoji icons per type)  
function formatNotificationText(notification) {  
 const { type, payload, actor\_email } = notification;  
 const actorName = actor\_email ? actor\_email.split('@')[0] : 'Someone';  
 switch(type) {  
 case "ball\_handoff":  
 return `🏀 Task handed to ${payload.toDepartment}: ${payload.title}`;  
 case "comment\_added":  
 return `💬 ${actorName} commented on: ${payload.title}`;  
 case "status\_changed":  
 return `📊 Task "${payload.title}" changed to ${payload.newStatus}`;  
 case "task\_created":  
 return `✨ New task created: ${payload.title || 'Untitled'}`;  
 case "task\_assigned":  
 return `👤 Task assigned to you: ${payload.title || 'Untitled'}`;  
 case "task\_overdue":  
 return `⏰ Task overdue: ${payload.title || 'Untitled'}`;  
 default:  
 return payload.title || type || "Notification";  
 }  
}

## Minimal Dashboard View

1. **Create a new Dashboard page:** Add a **DashboardPage.jsx** in the pages directory:

import { useEffect, useRef } from "react";  
import { useQuery } from "@tanstack/react-query";  
import { Chart } from "chart.js/auto";  
import { apiService } from "../services/api";  
  
export default function DashboardPage() {  
 const chartRef = useRef(null);  
  
 // Fetch task counts by status and by assignee  
 const { data: statusCounts = [] } = useQuery({  
 queryKey: ["dash\_status"],  
 queryFn: () => apiService.getTasksByStatus().then(res => res.data),  
 });  
 const { data: assigneeCounts = [] } = useQuery({  
 queryKey: ["dash\_assignee"],  
 queryFn: () => apiService.getTasksByAssignee().then(res => res.data),  
 });  
  
 // Initialize bar chart when statusCounts data is loaded  
 useEffect(() => {  
 if (!chartRef.current || statusCounts.length === 0) return;  
 const ctx = chartRef.current.getContext("2d");  
 const chart = new Chart(ctx, {  
 type: "bar",  
 data: {  
 labels: statusCounts.map(item => item.status),  
 datasets: [{  
 label: "Tasks by Status",  
 data: statusCounts.map(item => item.count),  
 backgroundColor: "#4C51BF"  
 }]  
 },  
 options: { responsive: true, maintainAspectRatio: false }  
 });  
 return () => chart.destroy();  
 }, [statusCounts]);  
  
 return (  
 <div className="mx-auto max-w-4xl p-4 space-y-6">  
 <h1 className="text-2xl font-semibold">Dashboard</h1>  
  
 {/\* Bar Chart: Task count per status \*/}  
 <div className="bg-white p-4 border rounded shadow-sm">  
 <h2 className="text-lg font-medium mb-3">Task Count by Status</h2>  
 <div style={{ height: "300px" }}>  
 <canvas ref={chartRef}></canvas>  
 </div>  
 </div>  
  
 {/\* Table: Task count per assignee \*/}  
 <div className="bg-white p-4 border rounded shadow-sm">  
 <h2 className="text-lg font-medium mb-3">Task Count by Assignee</h2>  
 <table className="w-full text-sm">  
 <thead>  
 <tr className="text-left border-b">  
 <th className="py-2">Assignee</th>  
 <th className="py-2">Tasks</th>  
 </tr>  
 </thead>  
 <tbody>  
 {assigneeCounts.map(row => (  
 <tr key={row.assignee || "none"}>  
 <td className="py-1">{row.assignee || "Unassigned"}</td>  
 <td className="py-1">{row.count}</td>  
 </tr>  
 ))}  
 </tbody>  
 </table>  
 </div>  
 </div>  
 );  
}

1. **Register the Dashboard route and navigation:** In **App.jsx**, import the new page and add a Dashboard tab and route:

/\* Import the DashboardPage component at the top of App.jsx \*/  
import DashboardPage from "./pages/DashboardPage";  
...  
function AppContent() {  
 return (  
 <>  
 {/\* Tabs Navigation Bar \*/}  
 <div className="tabs flex-wrap">  
 <NavLink to="/" className={({ isActive }) => `tab ${isActive ? 'active' : ''}`}>Projects</NavLink>  
 <NavLink to="/alltasks" className={({ isActive }) => `tab ${isActive ? 'active' : ''}`}>All Tasks</NavLink>  
 <NavLink to="/dashboard" className={({ isActive }) => `tab ${isActive ? 'active' : ''}`}>Dashboard</NavLink>  
 <NavLink to="/reports" className={({ isActive }) => `tab ${isActive ? 'active' : ''}`}>Reports</NavLink>  
 </div>  
 ...  
 {/\* Routes configuration \*/}  
 <Routes>  
 <Route path="/" element={<ProjectList ... />} />  
 <Route path="/alltasks" element={<AllTasksView />} />  
 <Route path="/dashboard" element={<DashboardPage />} />  
 <Route path="/reports" element={<Reports />} />  
 {/\* ...other routes... \*/}  
 </Routes>  
 </>  
 );  
}

## Quick Filters for Task Navigation

Add quick-filter buttons for "My Tasks" and "Ball in My Court" to the **AllTasksView.jsx** so users can filter the task list with one click:

import React, { useState } from "react";  
import { useQueryState } from "../hooks/useQueryState";  
import { useTasksQuery } from "../hooks/useTasksQuery";  
import { devAuth } from "../services/api";  
import TasksFilters from "./TasksFilters";  
import TaskItem from "./TaskItem";  
import CreateTaskModal from "./CreateTaskModal";  
  
export default function AllTasksView() {  
 const { getAll, set } = useQueryState();  
 const qp = getAll();  
 const [isCreateModalOpen, setIsCreateModalOpen] = useState(false);  
  
 // Fetch filtered tasks based on query params  
 const { data, loading, error } = useTasksQuery({  
 status: qp.status,  
 priority: qp.priority,  
 assignee: qp.assignee,  
 project: qp.project,  
 department: qp.department,  
 bic: qp.bic,  
 q: qp.q,  
 overdue: qp.overdue,  
 idle: qp.idle,  
 page: qp.page || "1",  
 limit: qp.limit || "20",  
 sort: qp.sort || "updated\_at:desc"  
 });  
  
 // Current user ID for quick filters  
 const myId = devAuth.getCurrentUser().id;  
  
 return (  
 <div className="space-y-4">  
 {/\* Filter bar and Create button \*/}  
 <div className="flex flex-col sm:flex-row items-start sm:items-center justify-between">  
 <div className="flex-1">  
 <TasksFilters />  
 </div>  
 <button  
 className="px-4 py-2 rounded bg-black text-white hover:bg-gray-800 mt-3 sm:mt-0 sm:ml-4"  
 onClick={() => setIsCreateModalOpen(true)}  
 >  
 + Create Task  
 </button>  
 </div>  
  
 <CreateTaskModal   
 isOpen={isCreateModalOpen}   
 onClose={() => setIsCreateModalOpen(false)}   
 />  
  
 {/\* Quick filter tabs \*/}  
 <div className="flex gap-2">  
 <button  
 className={`btn text-sm px-3 py-1 ${qp.assignee === myId ? "bg-gray-200 font-medium" : ""}`}  
 onClick={() => set({ assignee: myId, bic: "", page: "1" })}  
 >  
 My Tasks  
 </button>  
 <button  
 className={`btn text-sm px-3 py-1 ${qp.bic === myId ? "bg-gray-200 font-medium" : ""}`}  
 onClick={() => set({ bic: myId, assignee: "", page: "1" })}  
 >  
 Ball in My Court  
 </button>  
 </div>  
  
 {/\* Task list results \*/}  
 <div className="card p-5">  
 <div className="flex items-center justify-between mb-4">  
 <div className="text-sm text-gray-500">  
 {data ? `${data.total} task${data.total !== 1 ? "s" : ""} found` : "All Tasks"}  
 </div>  
 {data && data.totalPages > 1 && (  
 <div className="text-sm text-gray-500">  
 Page {data.page} of {data.totalPages}  
 </div>  
 )}  
 </div>  
  
 {loading && <div className="text-sm text-gray-500">Loading tasks...</div>}  
 {error && <div className="text-sm text-red-600">Error: {error}</div>}  
  
 {!loading && !error && data && (  
 <div className="grid gap-3">  
 {data.items.length === 0 ? (  
 <div className="text-sm text-gray-500">No tasks found. Try adjusting your filters.</div>  
 ) : (  
 data.items.map(t => <TaskItem key={t.id} t={t} />)  
 )}  
 </div>  
 )}  
  
 {/\* Pagination controls \*/}  
 {!loading && !error && data && data.totalPages > 1 && (  
 <div className="flex items-center justify-center gap-2 mt-4">  
 <button  
 className="btn text-sm px-3 py-1"  
 disabled={data.page === 1}  
 onClick={() => set({ page: String(data.page - 1) })}  
 >  
 Previous  
 </button>  
 <span className="text-sm text-gray-600">  
 Page {data.page} of {data.totalPages}  
 </span>  
 <button  
 className="btn text-sm px-3 py-1"  
 disabled={data.page === data.totalPages}  
 onClick={() => set({ page: String(data.page + 1) })}  
 >  
 Next  
 </button>  
 </div>  
 )}  
 </div>  
 </div>  
 );  
}

## Mobile Responsiveness Tweaks

Make sure the UI adjusts on small screens by applying these minor updates:

* **AllTasksView.jsx:** Already updated above – the filter bar now stacks on mobile (flex-col) and the Create button has mt-3 sm:mt-0 for spacing when stacked.
* **TaskItem.jsx:** Allow the task header to wrap on narrow screens. Update the header <div> to include flex-wrap:

<div className="flex items-start justify-between mb-2 flex-wrap">  
 <div className="flex-1">  
 <div className="flex items-center gap-2">  
 <h4 className="font-medium">{t.title}</h4>  
 {t.origin && (  
 <span className="text-xs px-2 py-0.5 rounded bg-blue-50 border border-blue-200 text-blue-700">  
 {originIcon} {t.origin}  
 </span>  
 )}  
 </div>  
 {t.description && (  
 <p className="text-sm text-gray-600 mt-1 line-clamp-2">{t.description}</p>  
 )}  
 </div>  
 <span className={`text-xs px-2 py-1 rounded ${statusColors[t.status] || 'bg-gray-100'}`}>  
 {t.status}  
 </span>  
</div>

* **TaskForm.jsx:** Change the two-column grids to single-column on small devices. For example, replace both instances of grid-cols-2 with grid-cols-1 md:grid-cols-2:

<div className="grid grid-cols-1 md:grid-cols-2 gap-4">  
 {/\* Status and Priority fields \*/}  
 ...  
</div>  
<div className="grid grid-cols-1 md:grid-cols-2 gap-4">  
 {/\* Project and Assignee fields \*/}  
 ...  
</div>

* **TaskList.jsx (project task list):** Allow action buttons to wrap. Update the actions container to use flex-wrap and spacing:

<div className="actions flex flex-wrap gap-2 mt-2">  
 <button className="btn btn-primary" onClick={() => handleHandoff(task)}>  
 Hand Off Ball  
 </button>  
 <button className="btn btn-secondary" onClick={() => handleUpdate(task)}>  
 Update Status  
 </button>  
 <button className="btn btn-danger" onClick={() => handleDelete(task.id)}>  
 Delete  
 </button>  
</div>

With these changes in place, your Coordination UI will have grouped notifications with icons, a new dashboard page with a bar chart and summary table, convenient quick filters for tasks, and improved mobile responsiveness. You can paste the above code blocks into your Replit project to apply the enhancements. Enjoy your updated ERP UI!