Stock Levels Dashboard

Overview

The stock dashboard is a component that displays the current stock levels of a chemical inventory. It is a dashboard that is used to monitor the stock levels of a chemical inventory.

Features

- Display the current stock levels of a chemical inventory
- Sort, filter, and search the chemical inventory by any column
- See item details when clicking on a row
- Responsive design
- Mobile friendly
- Animations
- Loading State
- Colour coded chemical groups
- Threshold alerts
- Stock distribution chart
- Drum counts

Code

Below is the code for our Chemical Inventory Dashboard. It's broken down into sections to make it easier to understand the different parts that make up this complex visualization system.

Imports and Dependencies - Libraries and tools used

It takes some planning and online research to find the right libraries and tools to use. By utilising the Lucide icons and Recharts library, we can quickly create a dashboard that is both functional and aesthetically pleasing, without needing to create the graph design from the ground up. Imagine telling a computer exactly how you want the graph to look with 1s and 0s. Instead, we can use the Recharts library and its individual components to create a graph that looks like this:

Inventory Management System

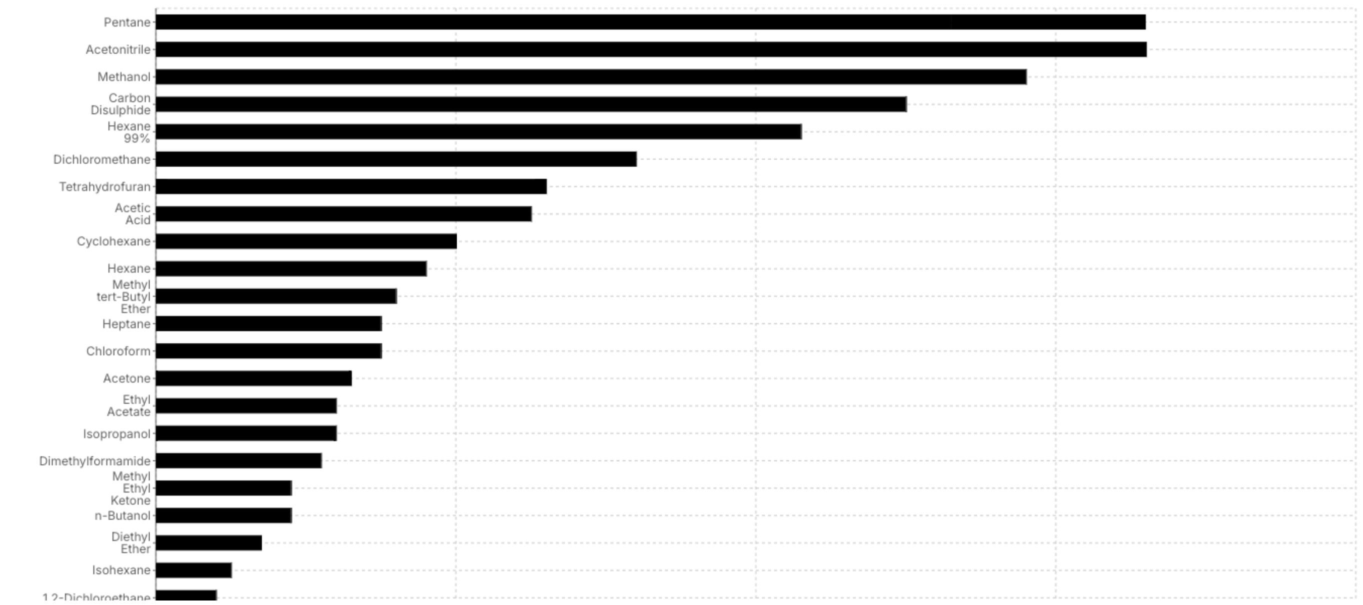
Chemical Solvent Inventory

Total Drums	New Drums	Repro Drums	Low Stock Alert
558	🟦 545	🟩 13	⚠ 78

Search by name, code or chemical group...

Sort by Name | Sort by Code | Sort by Total | Below Threshold | Collapse Chart

Solvent Drum Inventory



```
import React, { useState, useEffect } from "react";
import {
  BarChart,
  Bar,
  XAxis,
  YAxis,
  CartesianGrid,
  Tooltip,
  Legend,
  ResponsiveContainer,
} from "recharts";
import {
  ArrowUpDown,
  Filter,
  Search,
  AlertTriangle,
  Package,
  Maximize2,
  Minimize2,
} from "lucide-react";
import { selectFromTable } from "@/lib/database";
import { ViewType } from "@/types/models/base";
import { DrumInventory } from "../types";

/**
 * ChemicalInventoryDashboard is a React component that renders an interactive
 * dashboard for managing chemical solvent inventory. It fetches data from
 * a database, allows searching, sorting, and filtering of inventory items,
 * and displays a bar chart visualization of the stock levels. The component
 * also provides summary statistics and handles loading and error states.
 * Users can view detailed information about individual inventory items
 * through a conditional detail panel.
 */
```

Type Definitions and Color Configuration - Data structures and visual styling

```
// Chemical group type
type ChemicalGroup = "Hydrocarbons" | "Gen Solvents" | "Aromatics";

// Color mappings for different chemical groups
const newStockColors: Record<ChemicalGroup, string> = {
  Hydrocarbons: "#3b82f6", // blue
  "Gen Solvents": "#6366f1", // indigo
  Aromatics: "#8b5cf6", // purple
};

const reproStockColors: Record<ChemicalGroup, string> = {
  Hydrocarbons: "#10b981", // emerald
  "Gen Solvents": "#06b6d4", // cyan
  Aromatics: "#14b8a6", // teal
};
```

Helper Functions - Utility functions like color selection

```
// Helper function to determine bar color based on chemical group
const getBarColor = (chemGroup: string, isRepro: boolean): string => {
  const group = (chemGroup as ChemicalGroup) || "Gen Solvents";
  return isRepro
    ? reproStockColors[group as ChemicalGroup] ||
        reproStockColors["Gen Solvents"]
    : newStockColors[group as ChemicalGroup] || newStockColors["Gen Solvents"];
};
```

Main Component Declaration - The primary dashboard component

```
export default function ChemicalInventoryDashboard() {
```

State Management - Variables that control the dashboard's behavior

```
const [inventory, setInventory] = useState<DrumInventory[]>([]);
const [filteredInventory, setFilteredInventory] = useState<DrumInventory[]>([]);
const [loading, setLoading] = useState(true);
const [error, setError] = useState<string | null>(null);
const [searchTerm, setSearchTerm] = useState("");
const [sortConfig, setSortConfig] = useState({
  key: "name",
  direction: "asc",
});
const [selectedItem, setSelectedItem] = useState<DrumInventory | null>(null);
const [showLowStock, setShowLowStock] = useState(false);
const [isChartExpanded, setIsChartExpanded] = useState(false);
```

Data Calculations - Statistics and metrics derived from inventory data

```
// Calculate dynamic chart height based on number of items
const calculateChartHeight = () => {
  // Base height per item (adjust as needed for your design)
  const heightPerItem = 3; // vh units

  // Minimum height to ensure the chart is always visible
  const minHeight = 75; // vh

  // Calculate based on number of filtered items
  const calculatedHeight = Math.max(
    minHeight,
    filteredInventory.length * heightPerItem
  );

  return calculatedHeight;
};

// Get the calculated height
const dynamicChartHeight = calculateChartHeight();

// Fixed height value for each bar section
const barSectionHeight = "30px";

// Summary statistics
const totalNew = filteredInventory.reduce(
  (sum, item) => sum + item.newStock,
  0
);
const totalRepro = filteredInventory.reduce(
  (sum, item) => sum + (item.reproStock || 0),
  0
);
const totalStock = totalNew + totalRepro;
const lowStockCount = filteredInventory.filter(
  (item) => item.newStock + item.reproStock < (item.threshold || 0)
).length;
```

Data Fetching Logic - Code that retrieves inventory information

```
useEffect(() => {
  /**
   * Fetches inventory data from the database, transforms it for visualization,
   * and sets component state accordingly. Handles loading and error states.
   */
  async function fetchInventory() {
    try {
      setLoading(true);

      // Fetch inventory data from vw_drum_inventory view using our database layer
      const data =
        await selectFromTable<"vw_drum_inventory">("vw_drum_inventory");

      if (!data) throw new Error("No data returned from the database");

      // Transform data for visualization
      const transformedData = data.map((item) => ({
        id: item.code || "",
        code: item.code,
        name: item.value,
        newStock: item.raw_drums || 0,
        reproStock: item.repro_drums || 0,
        category: item.ch_group,
        chGroup: item.ch_group,
        threshold: item.threshold || 10,
        total: (item.raw_drums || 0) + (item.repro_drums || 0),
      }));

      setInventory(transformedData);
      setFilteredInventory(transformedData);
    } catch (err: any) {
      setError(err.message);
      console.error("Error fetching inventory:", err);
    } finally {
      setLoading(false);
    }
  }

  fetchInventory();
}, []);
```

Filtering and Sorting Logic - How the data is organized and

filtered

```
useEffect(() => {
  // Apply filters and search
  let result = [...inventory];

  if (searchTerm) {
    result = result.filter(
      (item) =>
        (item.name &&
          item.name.toLowerCase().includes(searchTerm.toLowerCase())) ||
        (item.code &&
          item.code.toLowerCase().includes(searchTerm.toLowerCase())) ||
        (item.category &&
          item.category.toLowerCase().includes(searchTerm.toLowerCase()))
    );
  }

  if (showLowStock) {
    result = result.filter(
      (item) => item.newStock + item.reproStock < (item.threshold || 0)
    );
  }

  if (!result || result.length === 0) return;

  // Apply sorting
  result.sort((a, b) => {
    if (
      a[sortConfig.key as keyof DrumInventory] != null &&
      b[sortConfig.key as keyof DrumInventory] != null &&
      a[sortConfig.key as keyof DrumInventory] <
        b[sortConfig.key as keyof DrumInventory]
    ) {
      return sortConfig.direction === "asc" ? -1 : 1;
    }
    if (
      a[sortConfig.key as keyof DrumInventory] != null &&
      b[sortConfig.key as keyof DrumInventory] != null &&
      a[sortConfig.key as keyof DrumInventory] >
        b[sortConfig.key as keyof DrumInventory]
    ) {
      return sortConfig.direction === "asc" ? 1 : -1;
    }
    return 0;
  });

  setFilteredInventory(result);
}, [inventory, searchTerm, sortConfig, showLowStock]);
```

Event Handlers - Functions that respond to user interactions

```
const handleSort = (key: string) => {
  setSortConfig((prevConfig) => ({
    key,
    direction:
      (prevConfig.key === key && prevConfig.direction === "asc") ||
      (key === "total" && prevConfig.key !== "total")
        ? "desc"
        : "asc",
  }));
};

const handleBarClick = (item: DrumInventory) => {
  setSelectedItem(item);
};
```

UI Rendering - The visual layout of the dashboard

```
// Loading state when page loads
if (loading)
  return (
    <div className="flex items-center justify-center h-64">
      Loading inventory data...
    </div>
  );
if (error) return <div className="text-red-500">Error: {error}</div>;

return (
  <div className="p-3 max-w-full bg-gray-50">
    {/* <h1 className="text-2xl font-bold mb-3">Inventory Management System</h1> */}
    <h2 className="text-xl font-semibold mb-4">Chemical Solvent Inventory</h2>

    {/* Summary Cards */}
    <div className="grid grid-cols-1 md:grid-cols-4 gap-3 mb-4">
      <div className="bg-white p-3 rounded-lg shadow">
        <h3 className="text-gray-500 text-sm">Total Drums</h3>
        <div className="flex items-center">
          <Package className="mr-2 text-blue-500" size={18} />
          <span className="text-2xl font-bold">{totalStock}</span>
        </div>
      </div>

      <div className="bg-white p-3 rounded-lg shadow">
        <h3 className="text-gray-500 text-sm">New Drums</h3>
        <div className="flex items-center">
          <div className="w-4 h-4 rounded mr-2 bg-blue-500"></div>
          <span className="text-2xl font-bold">{totalNew}</span>
        </div>
      </div>

      <div className="bg-white p-3 rounded-lg shadow">
        <h3 className="text-gray-500 text-sm">Repro Drums</h3>
        <div className="flex items-center">
          <div className="w-4 h-4 rounded mr-2 bg-green-500"></div>
          <span className="text-2xl font-bold">{totalRepro}</span>
        </div>
      </div>

      <div className="bg-white p-3 rounded-lg shadow">
        <h3 className="text-gray-500 text-sm">Low Stock Alert</h3>
        <div className="flex items-center">
          <AlertTriangle className="mr-2 text-amber-500" size={18} />
          <span className="text-2xl font-bold">{lowStockCount}</span>
        </div>
      </div>
    </div>

    {/* Controls */}
    <div className="flex flex-col md:flex-row gap-3 mb-4">
      <div className="flex items-center bg-white rounded-lg shadow px-3 py-2 flex-grow"
        <Search className="text-gray-400 mr-2" size={16} />
        <input
          type="text"
          placeholder="Search by name, code or chemical group..."
          className="flex-grow focus:outline-none text-sm"
          value={searchTerm}
          onChange={(e) => setSearchTerm(e.target.value)}
        />
      </div>

      <div className="flex gap-2 flex-wrap">
        <button
          className="flex items-center bg-white rounded-lg shadow px-3 py-1.5 text-sm"
          onClick={() => handleSort("name")}
        >
          <ArrowUpDown size={16} className="mr-1.5 text-gray-500" />
          Sort by Name
        </button>

        <button
          className="flex items-center bg-white rounded-lg shadow px-3 py-1.5 text-sm"
          onClick={() => handleSort("code")}
        >
          <ArrowUpDown size={16} className="mr-1.5 text-gray-500" />
          Sort by Code
        </button>

        <button
          className="flex items-center bg-white rounded-lg shadow px-3 py-1.5 text-sm"
          onClick={() => handleSort("total")}
        >
          <ArrowUpDown size={16} className="mr-1.5 text-gray-500" />
```

```
          Sort by Total
        </button>

        <button
          className={`flex items-center rounded-lg shadow px-3 py-1.5 text-sm ${
            showLowStock ? "bg-amber-100" : "bg-white"
          }`}
          onClick={() => setShowLowStock(!showLowStock)}
        >
          <Filter size={16} className="mr-1.5 text-gray-500" />
          {showLowStock ? "All Items" : "Below Threshold"}
        </button>

        <button
          className="flex items-center bg-white rounded-lg shadow px-3 py-1.5 text-sm"
          onClick={() => setIsChartExpanded(!isChartExpanded)}
        >
          {isChartExpanded ? (
            <>
              <Minimize2 size={16} className="mr-1.5 text-gray-500" />
              Collapse Chart
            </>
          ) : (
            <>
              <Maximize2 size={16} className="mr-1.5 text-gray-500" />
              Expand Chart
            </>
          )}
        </button>
      </div>
    </div>

    {/* Main Chart Area */}
    <div className="bg-white p-4 rounded-lg shadow mb-4">
      <h2 className="text-lg font-semibold mb-3">Solvent Drum Inventory</h2>

      {filteredInventory.length === 0 ? (
        <div className="text-center py-16 text-gray-500">
          No matching inventory items found.
        </div>
      ) : (
        <div
          className={
```

```
      isChartExpanded ? "" : "h-[75vh] overflow-y-auto relative"
    }
  >
    <div
      style={{
        height: `${isChartExpanded ? dynamicChartHeight : 200}vh`,
      }}
    >
      <ResponsiveContainer width="100%" height="100%">
        <BarChart
          layout="vertical"
          data={filteredInventory}
          margin={{ top: 5, right: 30, left: 80, bottom: 5 }}
          barSize={15}
          barCategoryGap={1}
          onClick={(data) =>
            data && handleBarClick(data.activePayload?.[0]?.payload)
          }
        >
          <CartesianGrid strokeDasharray="3 3" />
          <XAxis
            type="number"
            label={{
              value: "Number of Drums",
              position: "insideBottom",
              offset: -5,
            }}
          />
          <YAxis
            type="category"
            dataKey="name"
            tick={{ fontSize: 12, dy: 0 }}
            tickFormatter={(value: string) => {
              const item = filteredInventory.find(
                (item) => item.name === value
              );
              return item ? value : value;
            }}
            width={70}
            interval={0}
            tickSize={3}
          />
          <Tooltip
          formatter={(value: string, name: string) => [
            value, // TODO: Fix logic. `name` is the chemical name, not the sta
            name === "newStock" ? "New Drums" : "Repro Drums",
          ]}
          labelFormatter={(label: string) => {
            const item = filteredInventory.find(
              (item) => item.name === label
            );
            return item
              ? `${item.code} - ${label} (${item.category})`
              : label;
          }}
        />
        <Legend
          wrapperStyle={{
            paddingTop: 15,
            marginTop: 15,
            borderTop: "1px solid #f0f0f0",
          }}
          layout="horizontal"
          verticalAlign="bottom"
          align="center"
          payload={[
            // Hydrocarbons
            {
              value: "New Hydrocarbons",
              type: "square",
              color: newStockColors["Hydrocarbons"],
              id: "newHydrocarbons",
            },
            {
              value: "Repro Hydrocarbons",
              type: "square",
              color: reproStockColors["Hydrocarbons"],
              id: "reproHydrocarbons",
            },
            // Gen Solvents
            {
              value: "New Gen Solvents",
              type: "square",
              color: newStockColors["Gen Solvents"],
              id: "newGeneralSolvents",
            },
    {
      value: "Repro Gen Solvents",
      type: "square",
      color: reproStockColors["Gen Solvents"],
      id: "reproGeneralSolvents",
    },
    // Aromatics
    {
      value: "New Aromatics",
      type: "square",
      color: newStockColors["Aromatics"],
      id: "newAromatics",
    },
    {
      value: "Repro Aromatics",
      type: "square",
      color: reproStockColors["Aromatics"],
      id: "reproAromatics",
    },
  ]}
/>
<Bar
  dataKey="newStock"
  stackId="a"
  name="New Drums"
  onClick={(data: DrumInventory) => handleBarClick(data)}
  minPointSize={1}
  isAnimationActive={true}
>
  {filteredInventory.map((entry, index) => (
    <rect
      key={`new-${index}`}
      x={0}
      y={0}
      width={0}
      height={0}
      fill={getBarColor(
        entry.chGroup || "Gen Solvents",
        false
      )}
    />
  ))}
</Bar>
```

```
              <Bar
                dataKey="reproStock"
                stackId="a"
                name="Repro Drums"
                onClick={(data: DrumInventory) => handleBarClick(data)}
                minPointSize={1}
                isAnimationActive={true}
              >
                {filteredInventory.map((entry, index) => (
                  <rect
                    key={`repro-${index}`}
                    x={0}
                    y={0}
                    width={0}
                    height={0}
                    fill={getBarColor(
                      entry.chGroup || "Gen Solvents",
                      true
                    )}
                  />
                ))}
              </Bar>
            </BarChart>
          </ResponsiveContainer>
        </div>
      </div>
    )}
  </div>

  {/* Detail Panel (conditionally rendered) */}
  {selectedItem && (
    <div className="bg-white p-4 rounded-lg shadow">
      <div className="flex justify-between items-start mb-3">
        <h2 className="text-lg font-semibold">
          {selectedItem.code} - {selectedItem.name}
        </h2>
        <button
          className="text-gray-500 hover:text-gray-700"
          onClick={() => setSelectedItem(null)}
        >
          Close
        </button>
      </div>
```

```
<div className="grid grid-cols-1 md:grid-cols-2 gap-4">
  <div>
    <h3 className="text-base font-medium mb-2">
      Inventory Information
    </h3>
    <dl className="grid grid-cols-2 gap-x-4 gap-y-1 text-sm">
      <dt className="text-gray-500">Material Code:</dt>
      <dd className="font-medium">{selectedItem.code}</dd>
      <dt className="text-gray-500">Chemical Group:</dt>
      <dd>{selectedItem.category}</dd>
      <dt className="text-gray-500">New Drums:</dt>
      <dd className="font-semibold">{selectedItem.newStock} drums</dd>
      <dt className="text-gray-500">Repro Drums:</dt>
      <dd className="font-semibold">
        {selectedItem.reproStock} drums
      </dd>
      <dt className="text-gray-500">Total:</dt>
      <dd className="font-semibold">
        {selectedItem.newStock + selectedItem.reproStock} drums
      </dd>
      <dt className="text-gray-500">Threshold Level:</dt>
      <dd
        className={
          selectedItem.total < (selectedItem.threshold || 0)
            ? "text-red-500 font-bold"
            : ""
        }
      >
        {selectedItem.threshold} drums
      </dd>
    </dl>

    {selectedItem.total < (selectedItem.threshold || 0) && (
      <div className="mt-3 p-2 bg-amber-50 border border-amber-200 rounded-md f
        <AlertTriangle className="text-amber-500 mr-2" size={16} />
        <span className="text-amber-700">
          Stock is below threshold level
        </span>
      </div>
    )}
  </div>
```

```
<div>
  <h3 className="text-base font-medium mb-2">Stock Distribution</h3>
  <div className="h-40">
    <ResponsiveContainer width="100%" height="100%">
      <BarChart
        data={[
          {
            name: selectedItem.name,
            newStock: selectedItem.newStock,
            reproStock: selectedItem.reproStock,
            threshold: selectedItem.threshold,
          },
        ]}
      >
        <CartesianGrid strokeDasharray="3 3" />
        <XAxis dataKey="name" />
        <YAxis />
        <Tooltip
          formatter={(value: string, name: string) => {
            if (name === "newStock") return [value, "New Drums"];
            if (name === "reproStock")
              return [value, "Repro Drums"];
            if (name === "threshold") return [value, "Threshold"];
            return [value, name];
          }}
        />
        <Legend />
        <Bar dataKey="newStock" name="New Drums" fill="#3b82f6" />
        <Bar
          dataKey="reproStock"
          name="Repro Drums"
          fill="#10b981"
        />
        {/* Adding a reference line for the threshold */}
        <Bar
          dataKey="threshold"
          name="Threshold"
          fill="transparent"
          strokeDasharray="5 5"
          stroke="#f59e0b"
        />
      </BarChart>
    </ResponsiveContainer>
```

```
            </div>
          </div>
        </div>
      </div>
    )}
  </div>
);
}
```