**CartsByUserGridUtility.jsx** works and why the grid filters/sorts cleanly.

**Data fetching (axios) & state**

* **State you manage**
  + rows: normalized carts to feed the grid
  + usersById: a Map(id → user) for O(1) lookups
  + loading / err: UI state
* **useEffect() flow**
  + Fetch up to 100 carts:

const cartsRes = await axios.get("https://dummyjson.com/carts?limit=100");

const carts = (cartsRes.data?.carts ?? []).filter(Boolean);

* + Extract unique userIds and fetch those users in parallel:

const uniqueUserIds = [...new Set(carts.map(c => c.userId))].filter(id => id != null);

const users = await Promise.all(uniqueUserIds.map(id =>

axios.get(`https://dummyjson.com/users/${id}`).then(r => r.data).catch(() => null)

));

const map = new Map(users.filter(Boolean).map(u => [u.id, u]));

setUsersById(map);

* + - This de-duplicates requests and builds a fast lookup map.
    - Any individual user fetch can fail (caught and ignored) without breaking the page.
  + **Normalize carts** into stable, filter/sort-friendly rows:

const normalized = carts.map(c => {

const rawDate = c?.date ?? null;

const parsedDate = rawDate ? new Date(rawDate) : null;

const u = c?.userId != null ? map.get(c.userId) : null;

const userName = u ? `${u.firstName} ${u.lastName}` : (c?.userId != null ? `User #${c.userId}` : "");

return { ...c, \_rawDate: rawDate, \_parsedDate: parsedDate, userName };

});

setRows(normalized);

* + - Adds \_parsedDate (a real Date) for robust date sorting.
    - **Materializes userName** as a string field so filtering works (key fix).

The cancelled flag prevents state updates if the component unmounts mid-fetch.

**Grid columns & why filtering works now**

* **Cart #** (button → detail page)

renderCell: (params) => (

<Button component={Link} to={`/cart/${params.row.id}`}> {params.row.id} </Button>

)

* + Clicking takes you to /cart/:id (your Cart Details page).
* **User** (avatar + name)

field: "userName", // the actual string in each row

filterable: true, sortable: true,

renderCell: (...) // shows avatar + name, but filtering uses `userName`

* + The **built-in “contains”** filter works because the column is bound to a **real field** (userName), not a computed valueGetter. This fixes the classic “filter shows nothing” issue.
* **Date**

field: "\_parsedDate",

type: "dateTime",

valueGetter: (p) => p?.row?.\_parsedDate ?? null,

valueFormatter: (p) => p?.value ? new Intl.DateTimeFormat(...).format(p.value) : "—",

sortComparator: (v1, v2) => ... // numeric compare on Date objects

* + We store a Date so sorting is accurate. The column **formats** it for display but sorts by the underlying value.
* **Numeric columns** (Products, Qty, Totals)

{ field: "total", type: "number", valueFormatter: p => currency(p?.value) }

{ field: "discountedTotal", type: "number", valueFormatter: p => currency(p?.value) }

* + Bind directly to numeric fields and **format** with a currency helper for readability.

**Currency helper**

const currency = (v) =>

typeof v === "number"

? new Intl.NumberFormat(undefined, { style: "currency", currency: "USD" }).format(v)

: v ?? "";

* Keeps formatting consistent and safe for undefined values.

**Grid setup & UX**

* **Toolbar & Quick Filter**

slots={{ toolbar: GridToolbar }}

slotProps={{ toolbar: { showQuickFilter: true, quickFilterProps: { debounceMs: 500 } } }}

* + Gives you a global search field. Column filters also work (e.g., on **User** with “contains”).
* **Initial state**

initialState={{

sorting: { sortModel: [{ field: "\_parsedDate", sort: "desc" }] },

pagination: { paginationModel: { pageSize: 10, page: 0 } },

}}

* + Defaults to newest carts first; paginates to 10 rows.
* **getRowId / selection**

getRowId={(r) => r.id}

disableRowSelectionOnClick

* + Stable keys; clicking buttons/links doesn’t toggle selection.
* **Loading, empty, and errors**
  + loading shows the grid’s spinner.
  + NoRows renders a friendly message if nothing comes back.
  + Errors surface via <Alert severity="error">…</Alert>.

**Why this pattern is reliable**

* We **materialize** filter/sort targets (userName, \_parsedDate) on each row instead of relying on valueGetter.
* We keep **display** (renderCell/valueFormatter) separate from **data** (actual field values). The grid can then filter/sort on solid, static data.
* User lookups are cached in a Map, and requests are deduped for performance.