CartsDetails.jsx is doing and why it works reliably.

**1) Fetching the cart (and user) with axios**

* In useEffect, you fetch the cart by cartId:

const cRes = await axios.get(`https://dummyjson.com/carts/${cartId}`);

const c = cRes.data ?? null;

* If the cart has a userId, you fetch that user too:

const uRes = await axios.get(`https://dummyjson.com/users/${c.userId}`);

* Results are stashed into component state (cart, user) and a loading flag is managed around the request. Errors are caught and surfaced via err.

Why this is nice:

* Axios automatically JSON-parses responses.
* The cancelled flag prevents setting state if the component unmounts mid-request.
* Errors don’t crash the page; they show an <Alert />.

**2) Normalizing & computing row data (so the grid always shows numbers)**

All the “make it bulletproof” logic lives in this useMemo:

const { rows, totals } = useMemo(() => {

const src = Array.isArray(cart?.products) ? cart.products.filter(Boolean) : [];

// …normalize numbers (price, quantity, totals), compute fallbacks…

// return { rows: computed, totals: { … } }

}, [cart]);

What it does:

* Takes cart.products and builds **materialized rows** with concrete numeric fields:
  + price — unit price (from API; if missing, derive as total / quantity)
  + quantity — number
  + total — line total (from API; else compute price \* quantity)
  + discountPercentage — number
  + discountedTotal — line total after discount (from API; else compute)
* Computes footer rollups:
  + totalQty — sum of quantities
  + subtotal — sum of line totals
  + discountedTotal — sum of discounted line totals
  + apiTotal/apiDiscountedTotal — straight from the cart, for comparison

Why this is nice:

* The grid renders real numbers even if the API omits a field.
* Sorting/filtering stays reliable because the columns bind to plain fields (not computed getters).
* The totals footer uses the exact same data, so it can’t drift.

**3) Formatting money safely**

const currency = (v) =>

typeof v === "number" && Number.isFinite(v)

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

: v == null ? "—" : String(v);

* Centralized formatter used by the grid cells and the footer.
* Always shows a fallback dash when a number is missing.

**4) Presenting data in the MUI DataGrid**

**Columns definition (productColumns)**:

* **Product**: custom renderCell shows thumbnail + linked title (/product/:id).
* **Price / Line Total / Disc Total**: we bind to the concrete numeric fields (price, total, discountedTotal) and use renderCell to format with currency(...). This is more reliable across DataGrid versions than valueFormatter.
* **Qty** and **Disc %**: numeric columns, right-aligned.

**The grid itself**:

<DataGrid

rows={rows}

columns={productColumns}

getRowId={(r) => r?.id ?? `${r?.title ?? "row"}-${Math.random().toString(36).slice(2)}`}

loading={loading}

disableRowSelectionOnClick

slots={{ toolbar: GridToolbar, footer: TotalsFooter, noRowsOverlay: NoRows }}

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

initialState={{

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

sorting: { sortModel: [{ field: "product", sort: "asc" }] },

}}

/>

* **Toolbar**: search (quick filter) out of the box.
* **getRowId**: stable keys; falls back if id were ever missing.
* **Loading**: shows built-in spinner while axios fetches.
* **NoRows**: friendly message when the cart has no products.

**5) Totals inside the grid footer**

The custom TotalsFooter is injected via slots.footer and uses a simple CSS grid to line values under the correct columns:

<Box sx={{ display: "grid", gridTemplateColumns: "minmax(240px, 1fr) 110px 90px 130px 100px 130px" }}>

{/\* [Product] [Price] [Qty] [Line Total] [Disc %] [Disc Total] \*/}

<Box>Totals</Box>

<Box /> // no aggregate for Price

<Box>{totals.totalQty}</Box>

<Box>{currency(totals.subtotal)}</Box>

<Box /> // no aggregate for Disc %

<Box>{currency(totals.discountedTotal)}</Box>

</Box>

* Keeps the summary visually aligned with the grid columns.
* Also shows API totals below for sanity checks.

**6) Page chrome & UX**

* **Header** with title + “Back to Carts Utility” button (router link).
* **Error alert** if the fetch fails.
* **Skeleton** placeholders while initial data loads.
* Everything wrapped in <Container> and <Card> for consistent spacing and theming.

**Why this pattern worked (and is worth reusing)**

* Materialize fields → the grid sees plain numbers (no timing issues with getters).
* Guarded renderers → no crashes on null/undefined.
* Derived totals & footer → consistent with what’s on screen.
* Simple axios fetch + robust loading/error states.