Interview Exercise — Budget → Estimate (Stateful + UI Spec)

Why this update?

Problem: If `overheadPct` is derived from values spread across multiple Budget sections, we cannot compute it when only one section (e.g., Labor) has been submitted, unless the backend remembers previous submissions.
Solution: Make the backend *stateful* using a simple JSON file as storage. Each section submit *updates* this store; the backend then recomputes `overheadPct` from the latest known values and returns a refreshed `budgetSummary`. The client stores the latest summary and remains consistent after every API call.

UI Structure (Required)

Layout Overview

```
+--------------------------------------------------------------------------------+
| LEFT PANEL (Level 1)               |  MAIN AREA                                 |
|------------------------------------|---------------------------------------------|
|  • Budget                          |  • When "Budget" is selected -> shows       |
|  • Estimate                        |    Sub-Left Panel with section forms        |
+------------------------------------+---------------------------------------------+
| SUB-LEFT PANEL (Level 2 - Budget)  |  CONTENT AREA                               |
|------------------------------------|---------------------------------------------|
|   Labor                           |  • Right side displays the selected form    |
|   Equipment                       |  • Submit button posts to backend           |
|   Subcontractor                   |  • On success: show per-section subtotal    |
|   Taxes                           |    and update Budget Summary cards          |
+--------------------------------------------------------------------------------+
```

Budget View (Forms & Cards)

- **Subsections (click to show form):** Labor, Equipment, Subcontractor, Taxes.
- **Each form** has a *Submit* that calls its endpoint and updates the server JSON store.
- **On the right/top:** show three read-only cards (synced from backend snapshot):
  - Overhead % (`budgetSummary.overheadPct`)
  - Target Margin % (`budgetSummary.targetMarginPct`)
  - Sales Tax % (`budgetSummary.salesTaxPct`)
- **Below cards:** show the active section's computed subtotal (read-only).

Estimate View (Accordion Table)

- **Header Cards (read-only):** Overhead %, Target Margin %, Sales Tax % (from latest snapshot).
- **Actions:** *Add Item* button opens a product list; selecting a product adds a new row.
- **Table Columns (minimum):**

```
Product | Qty | Base Cost | Breakeven | Margin% | Price | Tax% | Price w/
Tax | Profit
```

- **Accordion behavior:** If a row is *manufactured*, it expands to show its component list (name, qty, cost per unit, extended cost). No operations in this exercise.
- **Footer Totals:** Sum of Price, Tax, Price w/ Tax, Profit (each × Qty).

Data Store (JSON on Server)

File schema (`budget-store.json`)

```json
{
  "version": 3,
  "lastUpdated": "2025-11-11T09:30:00Z",
  "labor": {
    "hourlyRate": 0,
    "hours": 0,
    "crewSize": 0,
    "laborBurdenPct": 0,
    "laborDirect": 0,
    "laborBurden": 0,
    "laborSubtotal": 0
  },
  "equipment": {
    "rentalPerDay": 0,
    "days": 0,
    "fuelCost": 0,
    "maintenancePct": 0,
    "equipBase": 0,
    "equipMaintenance": 0,
    "equipmentSubtotal": 0
  },
  "subcontractor": {
    "subQuote": 0,
    "mgmtFeePct": 0,
    "subMgmtFee": 0,
    "subcontractorSubtotal": 0
  },
  "taxes": {
    "salesTaxPct": 0,
    "targetMarginPct": 0
  },
  "budgetSummary": {
    "overheadPct": 0,
    "targetMarginPct": 0,
    "salesTaxPct": 0
  }
}
```

How the Backend Recomputes `overheadPct`

$$
\begin{aligned}
\text{DirectBase} &= \text{laborDirect} + \text{equipBase} + \text{subQuote} \\
\text{OverheadPart} &= \text{laborBurden} + \text{equipMaintenance} + \text{subMgmtFee} \\
\text{overheadPct} &= \begin{cases} 0 & \text{if DirectBase} = 0 \\ 100 \times \dfrac{\text{OverheadPart}}{\text{DirectBase}} & \text{otherwise} \end{cases}
\end{aligned}
$$

`targetMarginPct` and `salesTaxPct` are taken from the `taxes` section (user inputs).

Consistency (Client ↔ Server)

1. Client sends last known `version` (optional).
2. Server reads → updates → recomputes → increments `version` → writes atomically → returns full snapshot.
3. Client overwrites its local cache with the server response (single source of truth = server file).

Endpoints (Full Snapshot Responses)

GET `/api/budget`

Response

```
{
  "version": 3,
  "lastUpdated": "2025-11-11T09:30:00Z",
  "labor": { "...": "see store schema" },
  "equipment": { "...": "see store schema" },
  "subcontractor": { "...": "see store schema" },
  "taxes": { "salesTaxPct": 18, "targetMarginPct": 25 },
  "budgetSummary": { "overheadPct": 10, "targetMarginPct": 25, "salesTaxPct": 18 }
}
```

POST `/api/budget/labor`

Request

```
{ "hourlyRate": 300, "hours": 8, "crewSize": 2, "laborBurdenPct": 20, "version": 2 }
```

Server computes

$$
\begin{aligned}
\text{laborDirect} &= 300 \times 8 \times 2 = 4800 \\
\text{laborBurden} &= 4800 \times 0.20 = 960 \\
\text{laborSubtotal} &= 5760
\end{aligned}
$$

Then recompute `overheadPct` from the whole store. **Response**

```
{
  "version": 3,
  "lastUpdated": "2025-11-11T09:30:00Z",
  "labor": {
    "hourlyRate": 300,
    "hours": 8,
    "crewSize": 2,
    "laborBurdenPct": 20,
    "laborDirect": 4800,
    "laborBurden": 960,
    "laborSubtotal": 5760
  },
  "budgetSummary": { "overheadPct": 20, "targetMarginPct": 0, "salesTaxPct": 0 }
}
```

POST `/api/budget/equipment`

Request

```
{ "rentalPerDay": 1200, "days": 2, "fuelCost": 300, "maintenancePct": 8, "version": 3
    }
```

Server computes equipBase = 1200×2+300 = 2700, equipMaintenance = 216, equipmentSubtotal = 2916. Recompute summary. **Response**

```
{
  "version": 4,
  "lastUpdated": "2025-11-11T09:31:00Z",
  "equipment": {
    "rentalPerDay": 1200,
    "days": 2,
    "fuelCost": 300,
    "maintenancePct": 8,
    "equipBase": 2700,
    "equipMaintenance": 216,
    "equipmentSubtotal": 2916
  },
  "budgetSummary": { "overheadPct": 16.0, "targetMarginPct": 0, "salesTaxPct": 0 }
}
```

POST `/api/budget/subcontractor`

Request

```
{ "subQuote": 15000, "mgmtFeePct": 10, "version": 4 }
```

Server computes subMgmtFee = 1500, subcontractorSubtotal = 16500. Recompute summary. **Response**

```
{
  "version": 5,
  "lastUpdated": "2025-11-11T09:32:00Z",
  "subcontractor": {
    "subQuote": 15000,
    "mgmtFeePct": 10,
    "subMgmtFee": 1500,
    "subcontractorSubtotal": 16500
  },
  "budgetSummary": { "overheadPct": 12.0, "targetMarginPct": 0, "salesTaxPct": 0 }
}
```

POST `/api/budget/taxes`

Request

```
{ "salesTaxPct": 18, "targetMarginPct": 25, "version": 5 }
```

Server stores these two inputs and recomputes summary (overhead unchanged). **Response**

```
{
  "version": 6,
  "lastUpdated": "2025-11-11T09:33:00Z",
  "taxes": { "salesTaxPct": 18, "targetMarginPct": 25 },
  "budgetSummary": { "overheadPct": 12.0, "targetMarginPct": 25, "salesTaxPct": 18 }
}
```

POST `/api/budget/reset` (optional)

Resets the store to zeros, `version`=1.

Products (No Operations) & Catalog API

GET `/api/config`

```
{
  "budgetDefaults": { "overheadPct": 10, "targetMarginPct": 25, "salesTaxPct": 18 },
  "products": [
    {
      "id": "LED-IC144-W",
      "sku": "LED-IC144-W",
      "name": "LED Icicle 144 LEDs",
      "unit": "pcs",
      "isManufacturing": false,
      "cost": 450
    },
    {
      "id": "ICICLE-SET-3",
      "sku": "ICICLE-SET-3",
      "name": "Icicle Set (3 chains)",
      "unit": "set",
      "isManufacturing": true,
      "manufacturing": {
        "product": [
          { "id": "LED-IC144-W", "name": "LED Icicle 144", "sku": "LED-IC144-W", "qty":
               3, "cost": 450 }
        ]
      }
    }
  ]
}
```

Estimate Computation (Clear & Final)

Let $OH = \dfrac{\text{budgetSummary.overheadPct}}{100}$, $M = \dfrac{\text{budgetSummary.targetMarginPct}}{100}$, $T = \dfrac{\text{budgetSummary.salesT}}{100}$

1) Base Cost

- Simple: baseCost = product.cost.
- Manufactured: $\text{baseCost} = \sum_i (\text{comp}_i.\text{cost} \times \text{comp}_i.\text{qty})$.

2) Breakeven

$BE = \text{baseCost} \times (1 + OH)$.

3) Price before Tax (margin on price)

$\text{price} = \dfrac{BE}{1 - M}$.

4) Tax & Final

$\text{tax} = \text{price} \times T$, $\text{priceWithTax} = \text{price} + \text{tax}$.

5) Profit

$\text{profit} = \text{price} - \text{baseCost} - (\text{baseCost} \times OH)$.

Example (using the stateful numbers)

After the four submits above, summary is: $\{\text{overheadPct} = 12.0, \text{targetMarginPct} = 25, \text{salesTaxPct} = 18\}$.
Add `ICICLE-SET-3` (components only: 3×450).

$$
\begin{aligned}
\text{baseCost} &= 3 \times 450 = 1350 \\
OH &= 0.12,\ M = 0.25,\ T = 0.18 \\
BE &= 1350 \times 1.12 = 1512.00 \\
\text{price} &= \frac{1512.00}{1 - 0.25} = \frac{1512.00}{0.75} = 2016.00 \\
\text{tax} &= 2016.00 \times 0.18 = 362.88 \\
\text{priceWithTax} &= 2378.88 \\
\text{profit} &= 2016.00 - 1350.00 - (1350.00 \times 0.12) \\
&= 2016.00 - 1350.00 - 162.00 = 504.00
\end{aligned}
$$

If quantity = 2, multiply price, tax, final, profit by 2.

Implementation Hints (Non-normative)

- Use atomic writes: write to a temp file then rename.
- Increment `version` on every successful update.
- Client always trusts server snapshot and replaces local cache.
- Round display values to 2 decimals, keep internal math in floats.