FULL ENGINEERING SPECIFICATION — RESTAURANT RECIPE COSTING SYSTEM

(All explanations, diagrams, logic, validation rules, recursion behavior, and item-specific conversion system included)

---

1. SYSTEM OVERVIEW

This system is a **multi-layer recipe costing platform** designed for restaurants with complex, nested preparation workflows. Unlike simple cost calculators, this system fully supports:

- Multi-step recipes (raw → base → prepped → final dish)
- Components reused across multiple dishes (e.g., sauce used in 12 dishes)
- Components that themselves derive from earlier components
- Full ingredient cost traceability (down to raw items)
- Full labor cost traceability (multiple roles, minutes, batch scaling)
- Yield normalization (batch cost → cost per gram)
- Unit conversion for inconsistent vendor packaging (gallon → g, each → g)
- Error prevention: cycle detection, missing conversions, invalid structure
- Recursion with caching for performance

The final result:
Every menu item has an accurate, deep-layer cost breakdown, including both ingredients and labor, without double-counting or compounding errors.

---

2. KEY CONCEPTS AND DEFINITIONS

2.1 Item Types

There is only **one** “items” table; items differ by flags:

Raw Items

- Purchased directly
- No recipe lines
- Must have purchase info (unit, quantity, cost)
- Example: “Soy Sauce (10 kg / $40)”

Prepped Items

- Produced internally from raw/prepped items
- May also be a menu item
- Has recipe lines
- Has yield (amount produced after cooking)

Menu Items

- These are prepped items with `is_menu_item = true`
- Costed exactly like a prepped item
- Only difference: shown on the “menu” in UI or reports

2.2 Recipe Graph

Every `prepped` or `menu` item forms a **directed graph** of dependencies:

```
Raw items → Prepped components → Other components → Final Menu Item
```

Example:

```
Soy Sauce ---\
Sugar -------\
               → Teriyaki Sauce (yield 8000 g)
Chicken Thigh ------------------\
                                 → Teriyaki Chicken (menu item)
Teriyaki Sauce -----------------/
```

The cost engine recursively travels **down the graph** until raw items are reached.

2.3 Recipe Line Types

Each line inside a recipe is either:

Ingredient Line

- child = item_id of ingredient/prepped item
- quantity + unit required

Labor Line

- minutes required
- optional labor role (affects cost)

Ingredient lines and labor lines NEVER mix in one row.

2.4 Yield & Cost Scaling

Prepped items produce batches.
Cost formula:

```
Total Batch Cost = Total Ingredient Cost + Total Labor Cost

Cost Per Gram = Total Batch Cost / Yield_grams
```

Any recipe using this item multiplies:

```
Quantity_grams_used × Cost Per Gram
```

2.5 Unit Conversion Strategy

All cost calculations are normalized to **grams**.

Two conversion mechanisms:

A. Generic mass units

- g → g
- kg → g
- oz → g
- lb → g

Mapped with a simple conversion table.

B. Item-specific conversions

For units like:

- gallon
- liter
- cup
- tablespoon
- “each”

Each item requires a row in `item_unit_profiles` defining:

`grams_per_source_unit`

Example:

item	unit	grams
Large Egg	each	55
Medium Egg	each	45
Fry Oil	gallon	3600

If the user lists a unit like “each” or “gallon” and no specific conversion exists → **error**.

This guarantees consistent, factual calculations.

---

3. DATA MODEL (FINAL)

This section defines the exact relational schema your vendor will build.

3.1 Table: `items`

Stores all raw ingredients, prepped components, and menu items.

Field	Type	Required	Description
id	PK	yes	Unique item ID
name	string	yes	Name of item
item_kind	enum('raw','prepped')	yes	Determines costing path
is_menu_item	boolean	yes	True for final dishes
purchase_unit	string	raw only	Unit purchased (kg, gallon, each)

purchase_quantity	number	raw only	Amount purchased
purchase_cost	number	raw only	Cost of purchase_quantity
yield_amount	number	prepped only	How much finished product is produced
yield_unit	string	prepped only	Usually 'g'
notes	text	no	Optional notes

3.2 Table: `recipe_lines`

Defines every detail of how a prepped/menu item is produced.

Field	Type	Description
id	PK	Unique line
parent_item_id	FK → items.id	The item being made
line_type	enum('ingredient','labor')	Branches rules
child_item_id	FK → items.id	Ingredient/prepped item used (ingredient only)
quantity	number	Ingredient quantity
unit	string	Ingredient unit
labor_role	string/nullable	Labor role (labor only)
minutes	number	Minutes of work (labor only)

Validation rules:

Ingredient lines require:

- `child_item_id`, `quantity`, `unit`
  Labor lines require:

- `minutes`

And they must NOT mix fields from the other type.

3.3 Table: `item_unit_profiles`

Defines per-item conversion from non-mass units to grams.

Field	Type	Description
id	PK	—
item_id	FK	Item this rule is for
source_unit	string	Unit being converted (each, gallon, cup)
grams_per_source_unit	number	Exact grams per that unit
notes	text	—

3.4 Table: `labor_roles` (optional but recommended)

Field	Type	Description
id	PK	—
name	string	e.g., "Prep Cook"
hourly_wage	number	$ per hour

3.5 Table: `unit_conversions`

Generic mass conversions.

from_unit	to_unit	multiplier_to_grams
g	g	1
kg	g	1000

lb	g	453.592
oz	g	28.3495

---

4. COST ENGINE — DETAILED BEHAVIOR

This section defines how cost is computed, including recursion, cycle detection, caching, validation, and error handling.

4.1 Overview of Cost Calculation

Costing an item requires:

1. Ingredient costs
2. Labor costs
3. Yield normalization
4. Recursive resolution of dependencies

4.2 Ingredient Cost Calculation

For each ingredient line:

1. Convert `quantity + unit` → grams.
    - If mass unit → generic conversion
    - Else → lookup in item_unit_profiles
2. Resolve ingredient cost per gram recursively.
3. Multiply cost_per_gram × grams_used.

4.3 Labor Cost Calculation

```
labor_cost = (minutes / 60) × hourly_wage
```

Labor may appear multiple times.

4.4 Raw Item Cost Per Gram

```
cost_per_gram = purchase_cost / (purchase_quantity converted to grams)
```

Example:

- 10 kg Soy Sauce @ $40
- cost per gram = 40 / (10 × 1000) = 0.004 per gram

4.5 Prepped Item Cost Per Gram

```
total_batch_cost = sum(ingredient_costs) + sum(labor_costs)
cost_per_gram = total_batch_cost / yield_grams
```

---

5. RECURSION MODEL

This is the critical section vendors must implement correctly.

5.1 Pseudocode

```
function getCost(item_id):
    if item_id in cache:
```

```
        return cache[item_id]

    if item_id in visited:
        error("Cycle detected")

    add item_id to visited

    item = items[item_id]

    if item.item_kind == 'raw':
        cost_per_gram = compute_raw_cost(item)
        cache[item_id] = cost_per_gram
        remove item_id from visited
        return cost_per_gram

    # Prepped item
    lines = recipe_lines where parent_item_id = item_id

    ingredient_cost = 0
    labor_cost = 0

    for line in lines:
        if line_type == 'ingredient':
            grams = convert_to_grams(line.child_item_id,
line.quantity, line.unit)
            child_cost_per_gram = getCost(line.child_item_id)
            ingredient_cost += grams × child_cost_per_gram

        if line_type == 'labor':
            role_wage = lookup_role_wage(line.labor_role)
            labor_cost += (line.minutes / 60) × role_wage

    total_batch_cost = ingredient_cost + labor_cost
    cost_per_gram = total_batch_cost / item.yield_grams

    cache[item_id] = cost_per_gram
    remove item_id from visited
    return cost_per_gram
```

5.2 Required Safeguards

A. Cycle Detection

Must throw a fatal error if:

```
A → B → C → A
```

B. Missing Unit Profile

If a recipe line uses a non-mass unit and there's no mapping:

```
ERROR: Missing unit profile for item X with unit 'each'.
```

C. Missing Yield for Prepped Items

If prepped item missing yield:

```
ERROR: Prepped item X has no yield defined.
```

D. Zero Division

If yield_amount = 0 → error.

---

6. VALIDATION RULES (STRICT)

Ingredient lines must have:

- child_item_id
- quantity
- unit

Labor lines must have:

- minutes

For raw items:

- purchase_cost > 0
- purchase_quantity > 0
- purchase_unit valid

For prepped items:

- yield_amount > 0
- yield_unit must be a mass unit
- must have at least one recipe line

Units:

- Only mass units can use generic converter
- Non-mass must have item-specific profile

IDs not names:

- Absolutely no name matching to determine dependencies.

---

7. SAMPLE WORKFLOW (REALISTIC)

Step 1 — Create raw items

- Soy Sauce (10 kg @ $40)

- Sugar (5 kg @ $10)
- Chicken Thigh (20 kg @ $60)

Step 2 — Create prepped item: Teriyaki Sauce

Recipe:

Ingredient	Qty	Unit
Soy Sauce	5	kg
Sugar	1	kg
Labor	20	minutes

Yield: 8000 g

Step 3 — Create menu item: Teriyaki Chicken

Recipe:

Ingredient	Qty	Unit
Chicken Thigh	150	g
Teriyaki Sauce	80	g
Labor	5 minutes	

Step 4 — System computes:

- Cost per gram soy
- Cost per gram sugar
- Batch cost teriyaki sauce
- Cost per gram teriyaki sauce
- Ingredient + labor cost teriyaki chicken
- → total dish cost

---

8. API CONTRACT (RECOMMENDED)

This mirrors what a SaaS developer would build.

8.1 `GET /items`

Returns all items.

8.2 `GET /items/{id}`

Returns item metadata.

8.3 `GET /items/{id}/recipe`

Returns recipe lines.

8.4 `GET /items/{id}/cost`

Performs full recursive cost breakdown.

Includes:

- ingredientCost
- laborCost
- totalCost
- per-ingredient breakdown
- labor breakdown

And returns explicit errors for:

- cycles

- missing unit profiles
- missing yields
- invalid recipe structures

8.5 `POST /items`

Create item.

8.6 `POST /recipe_lines`

Add/edit recipe lines.

8.7 `POST /item_unit_profiles`

Define item-specific unit → grams mapping.

---

9. CONCLUSION

This specification provides everything required for a vendor to:

- Build the data model
- Implement recursion with caching
- Prevent cycles
- Handle mass/volume/count unit conversions
- Compute accurate cost per dish
- Produce detailed breakdown reports
- Validate inputs

- Expose API endpoints

If you give this document to any senior engineer or software vendor, they will understand every requirement and be able to build it without confusion.