Task: AI-Assisted Logistics Cleanup & Reconciliation

Timebox: up to 5 hours
Environment: any programming language, command-line program, no UI/DB
Tooling: you may use GPT or other assistants — smart use is part of the test

Goal

Write a program that:

1. **Cleans and normalizes messy order data**
2. **Plans courier assignments** under real constraints
3. **Reconciles** the plan vs. the actual delivery log
4. Produces **deterministic outputs**, plus a short **AI usage note** explaining how you used GPT effectively

Input Files (JSON/CSV — keep the schema exactly)

`orders.json`

Array of orders (notice duplicates, mixed casing, punctuation, and date formats):


```
[
  {
    "orderId": " Ord-001 ",
    "city": "6th of October",
    "zoneHint": "6 October- El Montazah",
    "address": "6 Oct - El Montazh,, st. 12",
    "paymentType": "COD",
    "productType": "fragile",
    "weight": 2,
    "deadline": "2025-08-12 16:30"
  },
  {
    "orderId": "ord001",
    "city": "6 October",
    "zoneHint": "6 October-El Montazah",
    "address": "6th of Oct., El-Montazah st 12",
    "paymentType": "cod",
    "productType": "Fragile",
    "weight": "2",
    "deadline": "2025/08/12 16:30"
  },
  {
    "orderId": "ORD-002.",
    "city": "Giza",
    "zoneHint": "Dokki",
    "address": "12 Dokki St.",
    "paymentType": "Prepaid",
    "productType": "standard",
```

```
    "weight": 3,
    "deadline": "2025-08-12 18:00"
  }
]
```

`couriers.json`

```
[
  {
    "courierId": "Bosta",
    "zonesCovered": ["6th of October", "Giza"],
    "acceptsCOD": true,
    "exclusions": ["fragile"],
    "dailyCapacity": 3,
    "priority": 2
  },
  {
    "courierId": "Weevo",
    "zonesCovered": ["6th of October", "Dokki", "Giza", "6 October"],
    "acceptsCOD": true,
    "exclusions": [],
    "dailyCapacity": 4,
    "priority": 1
  },
  {
    "courierId": "SafeShip",
    "zonesCovered": ["Dokki", "Giza"],
    "acceptsCOD": false,
    "exclusions": ["fragile"],
    "dailyCapacity": 10,
    "priority": 3
  }
]
```

`zones.csv`

Canonical mapping to **normalize** city/zone variants (you must use this):

```
raw,canonical
"6 October","6th of October"
"6th of Oct.","6th of October"
"El Montazah","El Montazah"
"El-Montazah","El Montazah"
"El Montazh","El Montazah"
"Dokki","Dokki"
"Giza","Giza"
```

`log.csv`

Actual delivery scans: `orderId, courierId, deliveredAt` (same day, local time)

```
Ord-001,BOSTA,2025-08-12 16:31
ORD-002,Weevo,2025-08-12 17:10
ORD-999,Weevo,2025-08-12 12:00
```

Requirements

A) Normalize & de-duplicate orders

- Normalize **orderId**: trim, uppercase, strip non-alphanumerics at ends ⇒ e.g. `" Ord-001 "` and `"ord001"` → `ORD-001`.
- Normalize **city** and **zoneHint** using `zones.csv` canonical values (case/typo tolerant).
- Coerce fields:
  - `paymentType` → one of `COD` or `Prepaid`
  - `productType` → lowercase canonical (e.g. `fragile`, `standard`)
  - `weight` → number
  - `deadline` → parsed datetime (accept both `YYYY-MM-DD HH:MM` and `YYYY/MM/DD HH:MM`)
- **Detect and merge duplicates** (same normalized `orderId`):
  - Prefer non-empty fields; for conflicting deadlines, keep the **earliest**.
  - If addresses clearly describe the same location (simple heuristic: normalized strings with distance/edit similarity), treat as one order; otherwise keep the earliest and add a **warning** describing the conflict.
- **Output** `clean_orders.json` with an optional `"warnings"` array for any dedupe/parse issues.

B) Plan courier assignments

For each unique cleaned order, assign **one** courier that:

- Covers the city/zone (match either normalized `city` or `zoneHint`).
- Satisfies constraints: `acceptsCOD`, and `productType` not in `exclusions`.
- Has enough **dailyCapacity** remaining (capacity measured by **sum of weights**).

Tie-breakers (in order):

1. Lower `priority` value (1 beats 2)
2. Tightest deadline (earlier deadline first)
3. Lowest current assigned load (by total weight)
4. Lexicographical `courierId`

Output `plan.json`:

```
{
  "assignments": [
    {"orderId": "ORD-001", "courierId": "Weevo"},
    {"orderId": "ORD-002", "courierId": "Weevo"}
  ],
  "unassigned": [
    {"orderId": "ORD-003", "reason": "no_supported_courier_or_capacity"}
  ],
  "capacityUsage": [
    {"courierId": "Bosta", "totalWeight": 0},
```

```
      {"courierId": "SafeShip", "totalWeight": 0},
      {"courierId": "Weevo", "totalWeight": 5}
    ]
  }
```

C) Reconcile plan vs. `log.csv`

Detect and output:

- **missing** — planned orders not present in the log
- **unexpected** — log orders not in `clean_orders.json`
- **duplicate** — same log order scanned > 1 time
- **late** — delivered after order `deadline`
- **misassigned** — delivered by a courier different from the **planned** one
- **overloadedCouriers** — any courier whose **actual delivered total weight** exceeds their capacity

Output `reconciliation.json`:

```
{
  "missing": [],
  "unexpected": [],
  "duplicate": [],
  "late": [],
  "misassigned": [],
  "overloadedCouriers": []
}
```

Determinism Requirements

- Sort all IDs and lists **alphabetically** in outputs.
- Parse both `YYYY-MM-DD HH:MM` and `YYYY/MM/DD HH:MM`.
- Normalize IDs to uppercase; normalize zones/cities via `zones.csv`.

Public Mini-Tests

Test 1 — Dedupe + Late + Unexpected + Misassigned

Use the sample `orders.json`, `couriers.json`, `zones.csv`, `log.csv` above.

Expected reconciliation highlights:

- `ORD-001` late by 1 minute (deadline 16:30; delivered 16:31).
- `ORD-001` should be **planned** to **Weevo** (Bosta excludes fragile).
- Log shows `ORD-001` delivered by **BOSTA** ⇒ **misassigned**.
- `ORD-002` on time by **Weevo**.
- `ORD-999` is **unexpected**.

- No courier exceeds capacity by actual weights.

Expected `reconciliation.json`:

```
{
  "missing": [],
  "unexpected": ["ORD-999"],
  "duplicate": [],
  "late": ["ORD-001"],
  "misassigned": ["ORD-001"],
  "overloadedCouriers": []
}
```

Test 2 — Capacity & Exclusions (planning)

`orders.json`:

```
[
  {
    "orderId": "A",
    "city": "Giza",
    "zoneHint": "Dokki",
    "address": "x",
    "paymentType": "COD",
    "productType": "fragile",
    "weight": 3,
    "deadline": "2025-08-12 18:00"
  },
  {
    "orderId": "B",
    "city": "Giza",
    "zoneHint": "Dokki",
    "address": "x",
    "paymentType": "COD",
    "productType": "standard",
    "weight": 2,
    "deadline": "2025-08-12 18:00"
  },
  {
    "orderId": "C",
    "city": "Giza",
    "zoneHint": "Dokki",
    "address": "x",
    "paymentType": "Prepaid",
    "productType": "standard",
    "weight": 6,
    "deadline": "2025-08-12 18:00"
  }
]
```

Use the same `couriers.json`.

Expected planning:

- `A` (fragile COD) → **Weevo** (Bosta/SafeShip exclude fragile; SafeShip also rejects COD).
- `B` (COD standard) → **Bosta** (accepts COD, supports Giza).
- `C` (prepaid standard) → **SafeShip**.

Expected `plan.json` (order of arrays may vary; keys sorted within each array is preferred):

```
{
  "assignments": [
    {"orderId": "A", "courierId": "Weevo"},
    {"orderId": "B", "courierId": "Bosta"},
    {"orderId": "C", "courierId": "SafeShip"}
  ],
  "unassigned": [],
  "capacityUsage": [
    {"courierId": "Bosta", "totalWeight": 2},
    {"courierId": "SafeShip", "totalWeight": 6},
    {"courierId": "Weevo", "totalWeight": 3}
  ]
}
```

Test 3 — Duplicate scans (reconciliation)

`log.csv`:

```
ORD-002,Weevo,2025-08-12 17:10
ORD-002,Weevo,2025-08-12 17:11
```

Expected `reconciliation.json`:

```
{
  "missing": [],
  "unexpected": [],
  "duplicate": ["ORD-002"],
  "late": [],
  "misassigned": [],
  "overloadedCouriers": []
}
```

Test 4 — Zone normalization

Orders with `"6 Oct"`, `"6th of Oct."`, `"6 October"` must normalize to `"6th of October"` and be assignable accordingly (no specific output provided; your `clean_orders.json` should reflect canonical zones).

Deliverables

1. **Source code** in a GitHub repo (any language).
   - A single entry point (e.g., `main.py` / `app.js`) that reads the four input files and writes the three outputs.
   - No external APIs, no UI/DB.
2. **README.md** — how to run (exact commands), dependencies, and assumptions.

3. **clean_orders.json** (may include `"warnings": [...]`)
4. **plan.json**
5. **reconciliation.json**
6. **ASSUMPTIONS.md** — normalization rules, dedupe heuristic, tie-breakers.
7. **AI_NOTES.md** (≤1 page) — 2–3 prompts you used with GPT, what you changed and why, and one thing GPT got wrong that you fixed.
8. *(Optional but nice)* `tests/` or a small script showing your own test cases.

Scoring (100)

- **Correctness (40):** matches behavior/outputs on public + hidden tests
- **Data Cleaning (20):** robust normalization, dedupe, parsing
- **Planning Logic (20):** constraints, tie-breakers, capacity usage
- **Reconciliation (10):** flags all categories, no false positives
- **Clarity & AI usage (10):** clear assumptions; thoughtful GPT usage (AI_NOTES shows discernment)

Rules

- Any language; no external APIs; local program only
- You may use standard parsing/date libraries
- Output must be **deterministic** (if using randomness, fix the seed)

Good luck.