Problem: "Optimal Truck Load Planner"

Problem Statement

Our logistics platform matches shippers with carriers. When a shipper creates a load (shipment), carriers bid on it.

A truck has limited **weight capacity** (e.g., 44,000 lbs) and **volume capacity** (e.g., 3,000 cubic feet) and can carry **multiple compatible orders** on the same route (multi-stop milk-run or LTL consolidation).

You need to build the core service that, given a truck and a list of available orders, returns the **optimal combination of orders** that:

- Maximizes revenue (sum of order.payout_to_carrier)
- Respects truck weight and volume limits
- All orders must be **compatible**:
    - Same origin → destination cities (or at least same headhaul lane)
    - Pickup and delivery windows must not conflict (we simplify: pickup date ≤ delivery date for all, and no overlapping time conflicts for now)

Requirements

One REST API (stateless, in-memory only)

POST /api/v1/load-optimizer/optimize
→ receives the JSON request below

→ returns the optimal combination of orders that maximizes payout_to_carrier while respecting weight, volume, hazmat, route, and time-window constraints.

Input (you will receive JSON via an API endpoint)
JSON

```
None
{
  "truck": {
    "id": "truck-123",
    "max_weight_lbs": 44000,
    "max_volume_cuft": 3000
  },
  "orders": [
    {
```

```
      "id": "ord-001",
      "payout_cents": 250000,          // $2,500.00
      "weight_lbs": 18000,
      "volume_cuft": 1200,
      "origin": "Los Angeles, CA",
      "destination": "Dallas, TX",
      "pickup_date": "2025-12-05",
      "delivery_date": "2025-12-09",
      "is_hazmat": false
    },
    {
      "id": "ord-002",
      "payout_cents": 180000,
      "weight_lbs": 12000,
      "volume_cuft": 900,
      "origin": "Los Angeles, CA",
      "destination": "Dallas, TX",
      "pickup_date": "2025-12-04",
      "delivery_date": "2025-12-10",
      "is_hazmat": false
    },
    {
      "id": "ord-003",
      "payout_cents": 320000,
      "weight_lbs": 30000,
      "volume_cuft": 1800,
      "origin": "Los Angeles, CA",
      "destination": "Dallas, TX",
      "pickup_date": "2025-12-06",
      "delivery_date": "2025-12-08",
      "is_hazmat": true
    }
    // ... up to 20 orders in real test
  ]
}
```

Expected Output

JSON

```
None
{
  "truck_id": "truck-123",
  "selected_order_ids": ["ord-001", "ord-002"],
  "total_payout_cents": 430000,
  "total_weight_lbs": 30000,
  "total_volume_cuft": 2100,
  "utilization_weight_percent": 68.18,
  "utilization_volume_percent": 70.0
}
```

Requirements (what we evaluate on)

1. Correctness – must never exceed weight/volume and must respect hazmat & route compatibility
2. Performance – must finish under 2 seconds even with 20–22 orders (2^{22} = 4M states → acceptable with DP + bitmask)
3. Code quality – clean, testable, well-named, SOLID
4. Real-world thinking:
    - Handle edge cases (empty orders, no feasible combination, etc.)
    - Use 64-bit integers for money (cents) – no float!
    - Thread-safety not required but bonus if you mention caching/memoization strategy
5. Bonus points (mention or implement any):
    - Dynamic Programming with bitmask (classic solution for n ≤ 22)
    - Recursive backtracking with pruning
    - Return all Pareto-optimal solutions (max revenue vs max utilization)
    - Configurable weights (e.g., prefer utilization over pure revenue)

Tech constraints & rules (important – please read)

- You may use any of these stacks: Java/Kotlin (Spring Boot, Micronaut, Quarkus), Go, Node.js/TypeScript (NestJS/Fastify/Express), Python (FastAPI), or C# (.NET 8).
- The service must be completely stateless → NO database allowed (no PostgreSQL, MySQL, SQLite, Mongo, etc.).
- In-memory caching (Caffeine, Guava, Redis, simple Map, etc.) is allowed and encouraged.
- Must run inside Docker (we will test with docker compose up).
- Must listen on port 8080.
- Must return correct HTTP status codes (400 on invalid input, 200 on success, 413 if payload too large, etc.).
- Money must be handled only in integer cents – never float/double.

What we will evaluate

1. Correctness on 10 hidden test cases (including n=22 orders, hazmat isolation, time-window conflicts, etc.)
2. Performance (< 800 ms on n=22 on our judge machine)
3. API design, validation, error handling
4. Code structure & readability
5. Dockerfile quality and README clarity

How to submit

1. Create a public GitHub repository and share the github link
2. Your repository must contain:
    - Complete source code
    - Dockerfile (multi-stage preferred)
    - docker-compose.yml that starts only your service (no DB needed)
    - README.md with exact instructions (see below)

README.md

````
None
# SmartLoad Optimization API

## How to run

```bash
git clone <your-repo>
cd <folder>
docker compose up --build
# → Service will be available at http://localhost:8080
````

Health check

```
None
curl http://localhost:8080/actuator/health   # or /healthz if you use Go/Node
```
```

Example request

None

```
curl -X POST http://localhost:8080/api/v1/load-optimizer/optimize \
  -H "Content-Type: application/json" \
  -d @sample-request.json
```