⚓ Backend Take-Home Test — Yard Planning

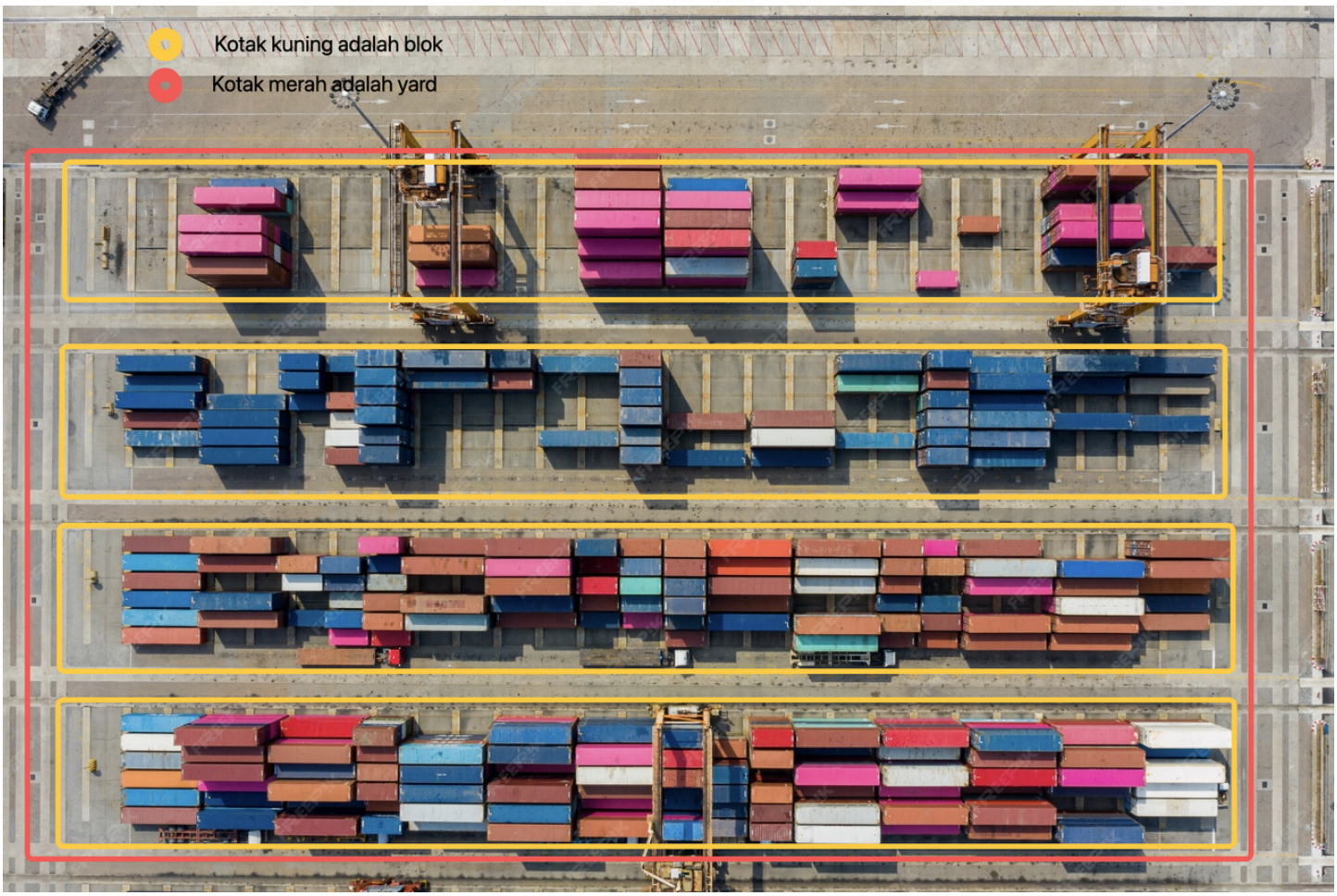
🚢 A. Konteks Pelabuhan

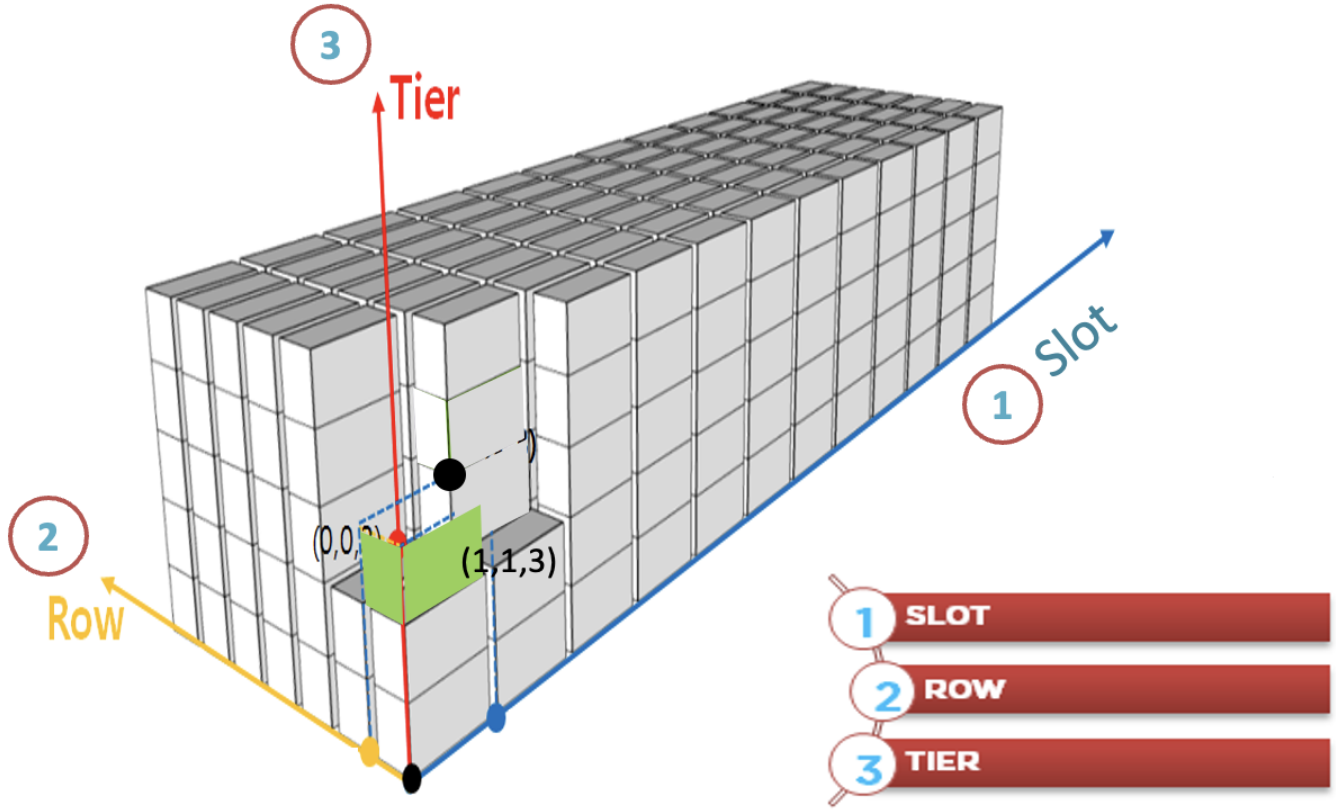
Pelabuhan merupakan area logistik tempat kapal melakukan aktivitas **bongkar muat kontainer**. Kontainer yang datang tidak langsung dikirim ke pelanggan, melainkan ditempatkan sementara di **lapangan penumpukan** (*yard*).

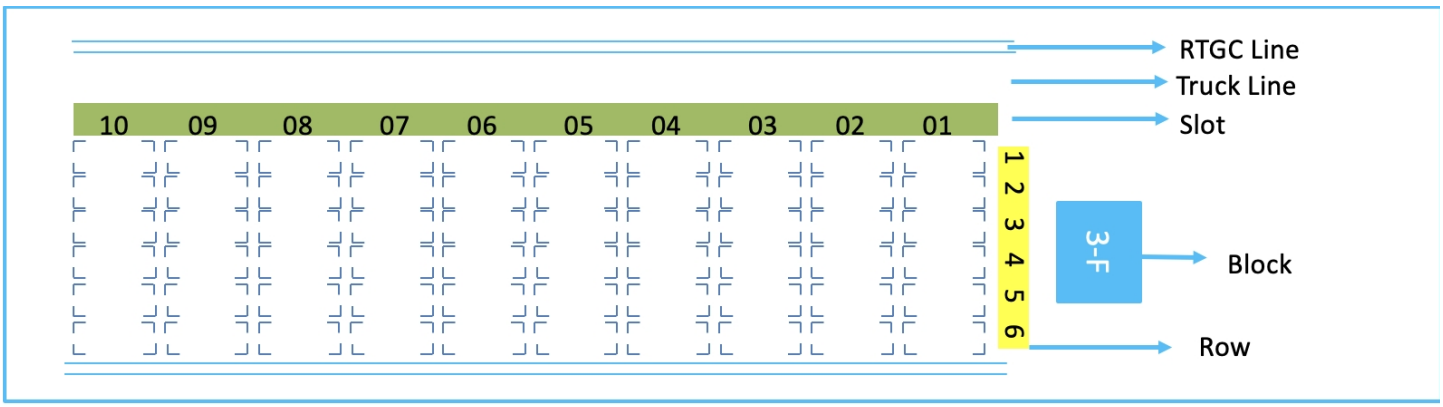
Sebuah *yard* dibagi menjadi beberapa **block**, dan setiap block memiliki struktur spasial:

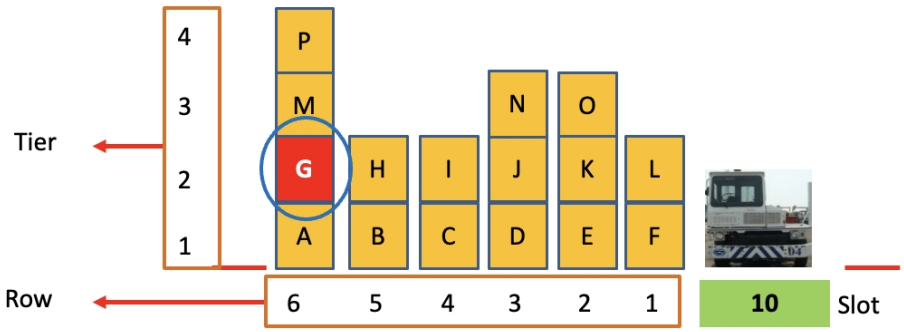
- **Slot** → sisi panjang block (horizontal)
- **Row** → sisi pendek block (vertikal)
- **Tier** → level tumpukan ke atas (vertikal)

Kombinasi `slot`, `row`, dan `tier` menentukan **posisi unik** kontainer di dalam yard.
Struktur ini digunakan oleh sistem untuk merencanakan **penempatan kontainer (Yard Planning)** — yaitu menentukan di mana setiap kontainer akan disimpan sesuai ukuran dan tipe-nya.









Yard Location:

Block-Slot-Row-Tier

Position of Container **"G"**
3F-10-6-2

⚙️ B. Studi Kasus

- Operator akan melakukan perencanaan suatu area block yang akan diisi spesifikasi container tertentu.
  - Container berukuran 20ft, Tinggi 8.6 dan tipe Dry akan direncanakan untuk slot 1-3 dan row 1-5

    - Container berukuran 40ft, Tinggi 8.6 dan tipe Dry akan direncanakan untuk slot 4-7 dan row 1-5
- Operator dapat meletakan container di lapangan pada area tertentu
- Operator dapat mengambil container di lapangan pada area tertentu

Catatan

> Container dengan ukurang 40ft membutuhkan 2 slot

🎯 C. Tujuan Test

Kamu diminta membangun **backend service** yang dapat:

1. Mengelola data *yard* dan *block*.
2. Melakukan **perencanaan penempatan kontainer** untuk area tertentu di block, berdasarkan spesifikasi ukuran dan tipe kontainer.
    - Operator dapat membuat rencana penempatan kontainer **untuk sebagian area block**, misalnya `slot 5-10` dan `row 1-2`.
    - Perencanaan bisa mempertimbangkan arah prioritas penumpukan (misalnya kiri→kanan, bawah→atas).
    - Setiap rencana hanya berlaku untuk kontainer dengan spesifikasi tertentu:
        - **Size:** 20ft atau 40ft
        - **Height:** 8.6ft atau 9.6ft
        - **Type:** Dry, Reefer, Open Top, dll
    - Sistem harus dapat:
        - Memastikan area yang direncanakan masih tersedia
        - Menghindari tumpang tindih antar rencana di area yang sama
3. Melakukan penempatan dan pengambilan container pada posisi tertentu

Kamu bebas menentukan:

- Arsitektur aplikasi
- Struktur tabel database
- Format input/output data

🧱 D. Cakupan Minimum Fitur

1️⃣ Design Database untuk yards, blocks, yard_plans

- Skema Database
  - Setiap yard memiliki beberapa block.
  - Block memiliki kapasitas: jumlah `slot`, `row`, dan `tier`.
- (**Optional**) CRUD Rest API

2️⃣ Rest API Development

Buat Restfull API untuk Simple Stacking System

1. Endpoint untuk mendapatkan saran posisi
   - Endpoint: POST /suggestion
   - Request Body

Request

```
{
  "yard": "YRD1"
  "container_number": "ALFI000001",
  "container_size": 20,
  "container_height": 8.6,
  "container_type": "DRY"
}
```

   - Response Body

Response

```
{
  "suggested_position": {
    "block": "LC01",
    "slot": 1,
    "row": 1,
    "tier": 1
  }
}
```

2. Endpoint untuk meletakan container di dalam yard
   - Endpoint: POST /placement
   - Request Body

Request

```
{
  "yard": "YRD1"
  "container_number": "ALFI000001",
  "block": "LC01",
  "slot": 1,
  "row": 1,
  "tier": 1
}
```

- Response Body

Respnse

```
{
  "message": "Success"
}
```

3. Endpoint untuk mengambil container dari lapangan
   - Endpoint: POST /pickup
   - Request Body

Request

```
{
  "yard": "YRD1"
  "container_number": "ALFI000001"
}
```

- Response Body

Response

```
{
  "message": "Success"
}
```

🧩 E. Persyaratan

Mandatory

- Bahasa pemrograman: **Go (Golang)**
- Database: **PostgreSQL**
- Validasi client input
- Tangani error dengan baik

Optional

- Terapkan concurrent execution
- Redis untuk caching
- Gunakan pendekatan modular (misalnya handler → service → repository).
- Boleh menggunakan `database/sql` atau `sqlx`.
- Gunakan linter
- Unit testing

🧠 F. Penilaian

Aspek	Bobot	Kriteria
Fungsionaliti	30%	Semua fungsi berjalan dengan baik dan benar
System Arsitektur & Clean Code	30%	System design, efficient recource, scalability, readability dan maintainability
Database Design	25%	Skema tabel efisien
Error Handling	10%	Penanganan error, validasi input, dan konflik data
Dokumentasi	5%	README yang menjelaskan desain & cara menjalankan

---

📦 G. Deliverables

1. Source code (repository GitHub/GitLab).
2. File `README.md` yang berisi:
    - Penjelasan desain sistem dan asumsi yang digunakan

    - Langkah untuk menjalankan aplikasi
3. (Opsional) Diagram ERD atau flow diagram sederhana.

🧭 H. Catatan Tambahan

- Test ini fokus pada **pemahaman domain dan kemampuan problem solving**, bukan banyaknya endpoint.
- Kandidat bebas menentukan model data dan struktur API.
- Gunakan nama field yang konsisten dengan konteks pelabuhan (yard, block, slot, row, tier).
- Solusi ideal bersifat fleksibel, agar nantinya dapat dikembangkan ke sistem yang lebih kompleks.