✅ Build a Coupon System MVP (in Go)

You are building an MVP for a **Coupon System** as part of a medicine ordering platform. Your task is to design and implement the backend logic using **Golang**, with a strong emphasis on correctness, modularity, and production-readiness.

📌 Core Requirements

1. Admin Coupon Creation

Create and manage coupon codes with:

- `coupon_code`: unique identifier
- `expiry_date`: expiration timestamp
- `usage_type`: `"one_time"`, `"multi_use"`, or `"time_based"`
- `applicable_medicine_ids`, `applicable_categories`
- `min_order_value`, `valid_time_window`
- `terms_and_conditions`, `discount_type`, `discount_value`, `max_usage_per_user`

2. Coupon Validation

- Validate against **all** constraints.
- Discount targets:
  - `inventory` (medicines)
  - `charges` (e.g., delivery fees)
- Enforce **correct usage type behavior** and handle **concurrent validations** correctly.

🔗 API Endpoints

> ⚠️ **Note:**
> The provided request/response JSON schemas below are **illustrative only**.
> You are encouraged to modify or extend these schemas as needed based on your system design.
> The goal is to **satisfy functional correctness** and **architectural clarity**, not to match the JSON formats exactly.

`GET /coupons/applicable`

Input:

```
{
  "cart_items": [
    { "id": "med_123", "category": "painkiller" }
  ],
  "order_total": 700,
  "timestamp": "2025-05-05T15:00:00Z"
}
```

Output:

```
{
  "applicable_coupons": [
    {
      "coupon_code": "SAVE20",
      "discount_value": 20
    }
  ]
}
```

`POST /coupons/validate`

Input:

```
{
  "coupon_code": "SAVE20",
  "cart_items": [...],
  "order_total": 700,
  "timestamp": "2025-05-05T15:00:00Z"
}
```

Success Output:

```
{
  "is_valid": true,
  "discount": {
```

```
    "items_discount": 50,
    "charges_discount": 20
  },
  "message": "coupon applied successfully"
}
```

Failure Output:

```
{
  "is_valid": false,
  "reason": "coupon expired or not applicable"
}
```

🧠 Technical Expectations

- **Concurrency-aware design** (goroutines, mutexes, or DB-level safety)
- **Persistent storage** (SQLite/Postgres)
- **Request-scoped context handling**
- **Caching** (LRU, TTL, MRU — any one)
- **OpenAPI docs** (Swagger UI or downloadable file)
- **Dockerized** and optionally deployed to a cloud URL

📦 Deliverables

- ✅ Code repo (public or zipped)
- ✅ `README.md` with:
  - Setup instructions
  - Architectural design
  - Concurrency/caching/locking notes
  - Swagger documentation link
  - Optional: Deployed API URL

For any **genuine** queries feel free to reach out to us at
vivek.kumar@farmako.in or sidharth.rathi@farmako.in