Unifize Engineering Manager Assignment (Backend)

Designing a Discount Platform

Context

You are an Engineering Manager at a large fashion e-commerce company. You are responsible for designing a Discount Platform that supports multiple discount types and will evolve over time.

Supported Discount Types

Payment offers	*10% instant discount on ICICI credit cards*
Brand discounts	*Min 40% off on all PUMA items*
Category discounts	*"Extra 10% off on all T-shirts"*
Voucher codes	SUPER69 : 69% off - with constraints (brand exclusions, customer tier, max discount cap)

Example Input

Cart

- 2 × PUMA T-shirts @ ₹999 each (Brand: PUMA, Category: T-shirts)
- 1 × Nike Shoes @ ₹4,999 (Brand: Nike, Category: Footwear)

Available Discounts

1. Brand-level: 40% off PUMA
2. Category-level: 10% off T-shirts (stackable)
3. Voucher: SUPER69
    - 69% off
    - Excludes Nike
    - Max cap ₹500
4. Payment offer:
    - 10% ICICI credit card discount
    - Max ₹200
    - Min cart value ₹2,000

Your system should

- Calculate the final payable price
- Clearly show which discounts were applied

Part 1: System Design (estimated 60 minutes)

README.md should have :

1. Canonical models
    a. Show 2–3 example rule definitions (JSON / pseudocode / DSL)
2. Constraints
    a. How do you handle conflicting discounts?
    b. How do you control the order of discounts applications?
    c. How do you enforce upper thresholds on discount?
3. Architecture diagram :
    a. Should service boundaries (what is separate vs combined),
    b. key APIs (high-level contracts only),
    c. data flow diagram among Cart,
    d. Pricing and Checkout and where discount rules live and how they are evaluated
    e. Hand-drawn diagram or Excalidraw / Whimsical/ Mermaid, etc diagram. We care about clarity, not diagram aesthetics. Boxes and arrows are enough.

Part 2: Implementation (estimated 90 minutes)

Implement only the core logic. Please don't implement databases, APIs, frameworks, or auth. In-memory data structures are enough. Proper separation of concerns and domain definitions are still expected.

Implement a component that calculates final pricing after applying multiple discounts.

```go
func CalculateDiscounts(
    cart Cart,
    discounts []Discount,
) DiscountResult

type DiscountResult struct {
    FinalPrice       Money
    AppliedDiscounts []AppliedDiscount
    Reasoning        string
}
```

Responsibility

- Apply eligible discounts in the correct order
- Calculate discount amounts and final payable price
- Explain why each discount was applied or skipped

General Guidance

- You may use *any* language, but Go or Java are preferred
- Prefer explicit domain types (Money, Cart, DiscountRule) over primitives
- Optimize for clarity and correctness, not clever abstractions
- Generate an initial implementation using an LLM (Cursor, Copilot, Claude, etc.)
- Treat it as a junior engineer's first PR
- Create *two* commits:
    1. Initial: AI-generated implementation
    2. Review: Improvements after human review
       (Explain what you fixed, removed, or redesigned — and why)

The goal is to evaluate your judgment, not your prompting skills.

Part 3: Testing Strategy (30 minutes)

No actual tests are expected

Briefly explain:

1. What MUST be unit tested (3–4 bullets)
2. What needs integration testing (2–3 scenarios)
3. What you would NOT test — and why over-testing is wasteful here
4. Include 1–2 pseudocode test cases showing edge-case thinking.

Submission Requirements

Repository Structure

```
discount-platform/
├── README.md
├── docs/
│   ├── architecture.(png | pdf | md)
│   ├── review-notes.md
│   └── testing-strategy.md
└── src/
    └── [your implementation]
```

FAQ

Can I use any language?
Yes. Feel free but do explain your choice if it is not your primary language professionally.

What if I disagree with the requirements?
Great. Document concerns and propose alternatives.

Is 3 hours really enough?
No — intentionally. Show prioritization and judgment.

What should I NOT build?

- Auth, APIs, databases, ORMs
- Admin tools or dashboards

- Deployment configs or Docker
- More than ~200 lines of implementation code