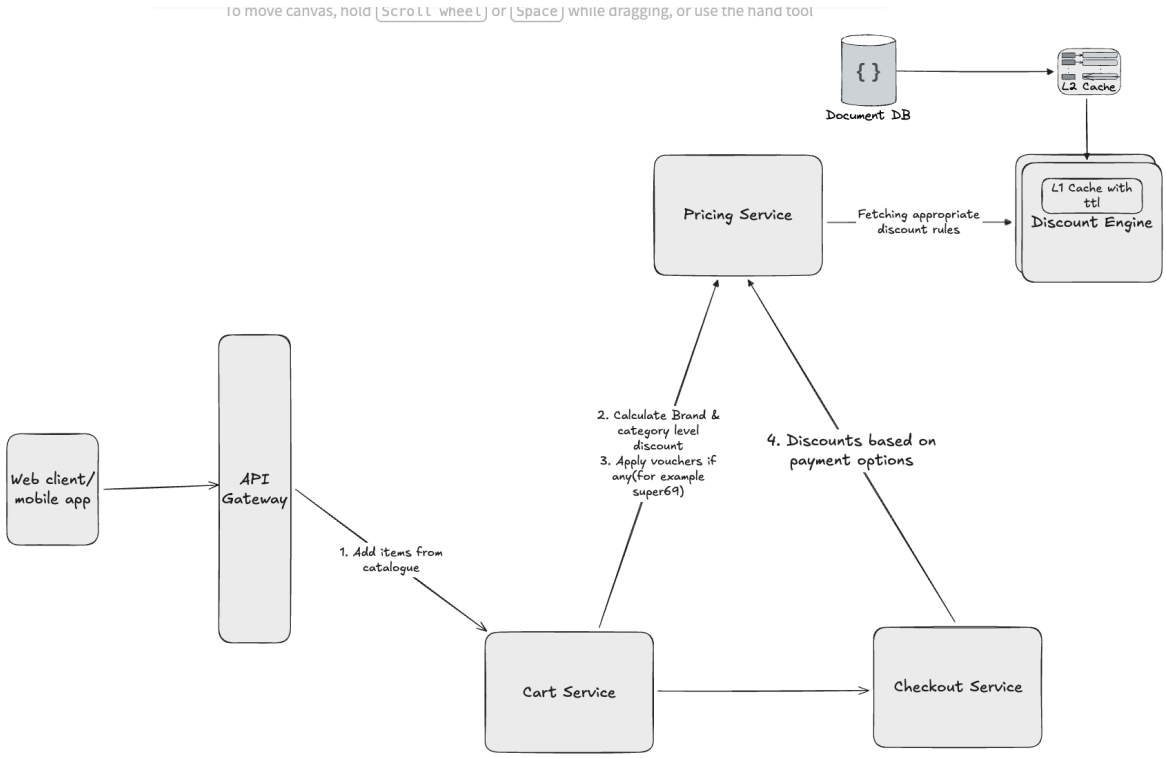
Ecommerce Discount Platform

1. High Level Design



2. Assumptions

a. Have not considered admin discount management api's for to focus on important functional aspects of discounting

3. Service components

a. **Web / Mobile Client**
   i. Owns user intent & interactions
   ii. Stateless
   iii. Never calculates prices or discounts

b. **API Gateway**
    i. Owns request routing, auth, rate limiting

c. **Cart Service**
    i. **Owns**
        1. Cart items, quantity
        2. Brand & category metadata per item
        3. Cart lifecycle (add/remove/update)
    ii. **Does NOT own**
        1. Price calculation
        2. Discount rules
        3. Payment logic

d. **Discount Engine**
    i. **Owns**
        1. Discount rule evaluation
        2. Eligibility checks
        3. Caps, exclusions, stackability
        4. Explaining *why* discounts applied or skipped
    ii. **Does NOT own**
        1. Cart state
        2. Checkout decisions
        3. Payment execution

e. **Rule Store (Document DB)**
    i. **Owns**
        1. Discount rule definitions
        2. Rule versioning & validity windows
    ii. **Does NOT own**
        1. Evaluation logic
        2. Runtime computation

f. **Checkout Service**
    i. **Owns**
        1. Payment selection
        2. Final order confirmation
        3. Persisting final price snapshot

    ii. **Does NOT own**

1. Discount calculation
2. Rule evaluation

4. Why Caching Is Needed Here?

a. Discount rules are read-heavy
b. Pricing is called on every cart update
c. Hitting DB per request is too slow and risky

5. L2 Cache (Shared Cache)(e.g. Redis)

What it stores?

- Active discount rules
- Indexed by type, brand, category, voucher code

Why

- Shared across all Pricing / Discount Engine instances
- Faster than DB
- Reduces DB load

When it updates

- On rule change Via async refresh / pub-sub
- On TTL expiry

6. L1 Cache (In-Memory, Per Instance Of Discount Engine)

What it stores

- Fully parsed rule objects
- Pre-compiled conditions

Why

- Ultra-low latency
- No network calls during pricing
- Ensures deterministic performance during traffic spikes

TTL

- Short (seconds–minutes)
- Safe to evict anytime

7. Discount Rule Store → Document DB (MongoDB / DynamoDB)

Stores

- Discount rule definitions
- Constraints & exclusions
- Priority & stackability
- Validity windows

Why

- Semi-structured JSON rules
- Schema evolves frequently

8. API Design

a. Calculate Discounts

Endpoint: `POST /api/v1/pricing/calculate`

Request Body:

```
{
  "cartId": "cart-123",
  "paymentMode": "credit_card",
  "paymentMethod": {
    "bank": "ICICI",
    "cardType": "CREDIT"
  }
}
```

Response Body:


```
{
  "originalTotal": 699700,
  "finalPrice": 537792,
  "currency": "INR",
  "appliedDiscounts": [
    {
      "discountId": "BRAND_PUMA_40",
      "type": "BRAND",
      "amount": 79920,
      "description": "40% off PUMA items"
    },
    {
      "discountId": "CAT_TSHIRT_10",
      "type": "CATEGORY",
      "amount": 11988,
      "description": "10% off T-shirts"
    }
  ],
  "skippedDiscounts": [
    {
      "discountId": "HDFC_15",
      "reason": "Payment method not HDFC"
    }
  ],
  "totalSavings": 161908
}
```


b. Validate Discount

Endpoint: `POST /api/v1/discounts/voucher/validate`

Request Body:


```
{
  "voucherCode": "SUPER69",
  "cartId": "cart-123",
  "customerId": "cust-456"
}
```

Response Body:

```
{
  "valid": true,
  "discountId": "SUPER69",
  "discountPercent": 69,
  "maxCap": 50000,
  "excludedBrands": ["Nike"],
  "estimatedSavings": 50000,
  "message": "Voucher applicable! Max savings ₹500"
}
```

c. Get Active Discounts

Endpoint: `GET /api/v1/discounts/active`

Response Body:

```
{
  "discounts": [
    {
      "id": "BRAND_PUMA_40",
      "type": "BRAND",
      "description": "40% off PUMA",
      "validUntil": "2024-12-31"
    }
  ],
  "count": 4
}
```