1.What is the “Claim Discount” flow?
It is a backend flow where a user claims a discount during a live flash sale. The system validates campaign status, inventory, and user eligibility, then reserves inventory and returns a discount token safely.

2.How do you ensure the flow is atomic?
We use MongoDB’s atomic conditional update ($inc with a condition). Inventory is incremented only if it is still available, ensuring no partial updates.

3.How do you ensure idempotency?
We enforce a unique constraint on (userId + campaignId).
If the same user retries or refreshes, the existing claim is returned and inventory is not deducted again.

4.How do you prevent overselling inventory?
Inventory is updated using an atomic condition:

- Increment inventory only if claimedInventory < totalInventory
- If inventory reaches zero, further claims fail

This guarantees inventory never goes negative.

5.How do you validate the campaign is LIVE?
We check:

- startTime <= currentTime
- endTime >= currentTime

Only live campaigns can accept claims.

6.How do you calculate the discount
Discount is calculated using campaign rules:

- Percentage of cart value
- Capped by maximum discount
- Applied only if cart value meets minimum requirement

7.How do you handle retries or page refresh
Retries return the same claim record because of idempotency.
No new inventory is reserved on retry.

8.How do you handle high concurrency (10,000 users)
MongoDB atomic updates and unique indexes handle concurrency safely.
Even if many users click at the same time, only valid claims succeed.

9.Why didn't you use Redis?
MongoDB atomic operations are sufficient for correctness.
Avoiding Redis keeps the system simpler while still preventing overselling.

10.How do you handle duplicate API calls?
Duplicate calls are handled by idempotency.
The backend detects an existing claim and returns the same response.

11.How do you ensure users cannot claim for others?
In production, user identity is derived from JWT, not request body.
The backend trusts only the authenticated user from the token.

12.How do you ensure claim tokens are secure?
Claim tokens are randomly generated and not user-controlled, making them non-forgeable.

13.How do you handle a user opening multiple tabs?

Answer:
The unique (userId + campaignId) constraint ensures only one claim per user, regardless of tabs.

14.How do you handle MongoDB failures or retries?
If a request fails, the client can retry safely because the operation is idempotent.

15.How is the system scalable?
The system is stateless, uses indexed queries, and relies on MongoDB's atomic operations, making it horizontally scalable.

16.What tradeoffs did you make?
We chose simplicity over additional infrastructure.
No Redis or Kafka was used to reduce complexity while maintaining correctness.

17.What would you change if Redis was allowed?
Redis could be used for:

- Faster idempotency checks
- Temporary inventory reservation
- Reduced database load

18.What would Kafka add?
Kafka would enable:

- Asynchronous processing
- Analytics events
- Audit logs
- Better observability