Case brief and take home

A1. Problem Statement

Build a minimal **Orders** microservice in **Go** that exposes:

- `POST /orders` – create an order
- `GET /orders/{id}` – retrieve by id
- `GET /healthz` – health check

A2. Functional/Non-Functional Requirements

- **Persistence:** PostgreSQL (schema of your choice).
- **Events:** On create, emit an `OrderCreated` event to an in-process queue (Go channel) and process it with a background worker (simulate email/notification).
- **Reliability:** Context propagation (`context.Context`), graceful shutdown, basic error taxonomy.
- **Quality:** Unit tests for core logic, minimal OpenAPI/Swagger, 12-factor config via env vars.
- **Ops:** Dockerfile; `docker-compose.yml` to run app + Postgres; migrations.
- **Nice to have:** structured logging (zap/zerolog), OTel trace skeleton, idempotency for `POST /orders`.

A3. Deliverables

- Git repo with code, tests, **README** (how to run), migration, `docker-compose.yaml`.
- Short **Design Notes** (~½ page): key decisions, trade-offs, "If I had 1 more day…".

A4. Timebox & Rules

- Max 2-ish hours. Aim for correctness over features.
- No external services beyond Postgres. Prefer stdlib + well-known libs.

A5. What we'll talk about

- Idiomatic Go, testing discipline, concurrency correctness.
- API hygiene, observability hooks, pragmatism.

Live Extension & Debugging (30-45 min)

B1. Agenda

1. **Candidate walkthrough** (10 min)
2. **Live coding** (25 min)
    - Add **idempotency** to `POST /orders` using `Idempotency-Key` header.
    - Ensure **context cancellation** flows to DB and worker.
3. **Debugging** (15 min)
    - Investigate a goroutine/worker bug (see E.2) and fix.

E.2. Debugging Snippet (goroutine leak/busy loop)

```
func startWorker(ctx context.Context, jobs <-chan Order, wg *sync.WaitGroup) {
wg.Add(1)
go func() {
defer wg.Done()
for {
select {
case <-ctx.Done():
return
case j := <-jobs: // BUG: missing ok check; spins on closed channel
process(j)
}
}
}()
}
```

Architecture & Kubernetes Deep Dive (45 min)

C1. Prompt

We are retiring a monolith and launching **Orders**, **Payments**, **Catalog** as microservices. Peak 2k RPS, EU data residency, zero-downtime deploys, operable by a lean team.

C2. Topics & Expected Coverage

- **Service boundaries & data ownership:** tables per service; avoid cross-service joins; read models.
- **Sync vs async:** REST for commands; **events** (Kafka/NATS) for propagation; idempotency; retries/backoff; DLQ.
- **Transactions/consistency:** outbox pattern or CDC; eventual consistency UX.
- **API evolution:** backward-compatible changes; versioning; feature flags.
- **Kubernetes manifests:** Deployment, Service, HPA, readiness/liveness probes, requests/limits, PDB, rolling updates; Helm vs Kustomize; GitOps.
- **Security:** mTLS/JWT between services, least-priv DB, secrets mgmt (External Secrets/Sealed Secrets), image scanning/SBOM.
- **Observability:** OTel traces across services, logs correlation, SLIs/SLOs (availability, latency p95), alerting philosophy.
- **Cost/scale:** autoscaling signals (CPU + Q length), cache, cold starts.

C3. Hands-On

- Draft a minimal **Deployment + Service + HPA** for Orders with env vars and Secret references.