Prueba Técnica – Senior Backend (Node.js + MySQL + Docker + Lambda)

Objetivo

Construir un sistema mínimo compuesto por dos APIs (Customers y Orders) y un Lambda orquestador. Las APIs deben operar sobre MySQL, estar documentadas y levantarse con Docker Compose. El Lambda se invoca desde Postman/Insomnia (local via serverless-offline + ngrok o desplegado en AWS) y orquesta la creación y confirmación de pedidos, devolviendo un JSON consolidado. **Entregables (Repositorio GitHub)**

1. Monorepo con carpetas: /customers-api, /orders-api, /lambda-orchestrator, /db (schema.sql y seed.sql), docker-compose.yml, README.md, openapi.yaml por servicio.
2. Ambas APIs y el Lambda en Node.js (JavaScript) + Express, autenticación JWT simple, validación (Zod/Joi), SQL parametrizado.
3. Lambda orquestador con Serverless Framework (runtime Node 22). Endpoint HTTP para orquestar pedido.
4. Documentación OpenAPI 3.0 por servicio + ejemplos cURL/collection Postman/Insomnia (opcional).
5. Scripts NPM: build, start/dev, migrate, seed, test (si agregas pruebas).

Caso de uso: Backoffice de Pedidos B2B (mínimo indispensable)

- Operador crea/gestiona clientes (Customers API).
- Operador gestiona productos y stock (Orders API).
- Operador crea un pedido para un cliente existente; el sistema valida al cliente en Customers API, verifica stock, calcula totales y crea la orden en estado CREATED.
- Operador confirma el pedido; el sistema cambia a CONFIRMED en forma idempotente (X-Idempotency-Key).

Requisitos funcionales por servicio

Customers API (puerto 3001)

- POST /customers → crea cliente {name, email (único), phone}.
- GET /customers/:id → detalle.
- GET /customers?search=&cursor;=&limit;= → búsqueda.

- PUT /customers/:id / DELETE /customers/:id (soft-delete opcional).
- GET /internal/customers/:id → igual al GET normal pero requiere Authorization: Bearer SERVICE_TOKEN (para Orders).

Orders API (puerto 3002)

- Productos: POST /products, PATCH /products/:id (precio/stock), GET /products/:id, GET /products?search=&cursor;=&limit;=.
- Órdenes:
- • POST /orders → body {customer_id, items:[{product_id, qty}]}; valida cliente en Customers (endpoint /internal), verifica stock, crea order (CREATED), descuenta stock (transacción).
- • GET /orders/:id → incluye items.
- • GET /orders?status=&from;=&to;=&cursor;=&limit;=.
- • POST /orders/:id/confirm → idempotente con header X-Idempotency-Key; devuelve mismo resultado si se repite la misma key.
- • POST /orders/:id/cancel → CREATED cancela y restaura stock; CONFIRMED cancela dentro de 10 min (regla simple).

Lambda Orquestador (HTTP)

Endpoint: POST /orchestrator/create-and-confirm-order. Recibe {customer_id, items[], idempotency_key, (opcional) correlation_id}. Flujo: valida cliente (Customers /internal), crea orden (Orders /orders), confirma orden (Orders /orders/:id/confirm con X-Idempotency-Key) y responde JSON consolidado (cliente + orden confirmada + items).

Ejemplo de request (JSON)

```
{ "customer_id": 1, "items": [ { "product_id": 2, "qty": 3 } ],
"idempotency_key": "abc-123", "correlation_id": "req-789" }
```

Ejemplo de response (201)

```
{ "success": true, "correlationId": "req-789", "data": { "customer": { "id": 1,
"name": "ACME", "email": "ops@acme.com", "phone": "..." }, "order": { "id": 101,
"status": "CONFIRMED", "total_cents": 459900, "items": [ { "product_id": 2, "qty":
3, "unit_price_cents": 129900, "subtotal_cents": 389700 } ] } } }
```

Base de datos (incluida en el repo)

- Tablas mínimas: customers(id, name, email único, phone, created_at), products(id, sku único, name, price_cents, stock, created_at),
- orders(id, customer_id FK, status ENUM('CREATED','CONFIRMED','CANCELED'), total_cents,

created_at),

- order_items(id, order_id FK, product_id FK, qty, unit_price_cents, subtotal_cents),
- idempotency_keys(key PK único, target_type, target_id, status, response_body, created_at, expires_at).
- Incluir /db/schema.sql y /db/seed.sql con datos de ejemplo.

Levantamiento local con Docker Compose

1. Clonar repo y copiar .env.example → .env en cada servicio.
2. docker-compose build
3. docker-compose up -d
4. Verificar: Customers http://localhost:3001/health, Orders http://localhost:3002/health.

Probar Lambda (local y AWS)

1. Local: serverless-offline en /lambda-orchestrator (npm i, npm run build, npm run dev).
2. Opcional: ngrok http 3000/3003 (según puerto offline) para URL pública.
3. AWS: serverless deploy (configurar CUSTOMERS_API_BASE y ORDERS_API_BASE con URLs públicas).
4. Invocar desde Postman/Insomnia al endpoint del Lambda con el JSON de ejemplo.

Criterios de aceptación mínimos

- Customers API y Orders API funcionales, documentadas (OpenAPI) y levantan con docker-compose.
- Creación de pedido: valida cliente, stock, calcula totales, crea CREATED y descuenta stock (transacción).
- Confirmación idempotente con X-Idempotency-Key; reintentos con misma key devuelven la misma respuesta.
- Cancelación válida restaura stock según reglas.
- Lambda orquestador invocable por HTTP y retorna JSON consolidado.
- Repo contiene /db/schema.sql y /db/seed.sql; README con pasos claros.

Entrega

Subir el repositorio a GitHub con todo lo indicado. En el README agregar: comandos para levantar, variables de entorno, URLs base, ejemplos cURL, y cómo invocar el Lambda en local y en AWS.