Blackbird

# Prueba técnica rol junior Backend developer Blackbird Labs

Este documento contiene el Bloque 1 (Etapa 1 y Etapa 2) con contexto operativo real y código para revisión. La prueba está diseñada para evaluar competencias técnicas.

**Bloque 1 - Etapa 1: Desarrollo API de Tracking con Checkpoints**

**Objetivo:** construir una API para registrar checkpoints y consultar el tracking de paquetes.

**Contexto ficticio y escala operativa**

- Un checkpoint es un toque/escaneo de una guía para mantener la trazabilidad y asegurar la carga durante todo un ciclo logístico.
- Un ciclo logístico tiene una **duración máxima** de 3 días por guía y se realizan en promedio al rededor de 10 checkpoints por guía (En promedio 3.3 checkpoints por día por guía). Cada guía transporta en promedio 1.22 unidades (Una unidad es parte de la carga, imagina el siguiente ejemplo: un cliente ha comprado 3 artículos, es posible que le envíen una guía con las 3 unidades, o 2 guías, una con 2 unidades y otra con la unidad restante, todo dependerá del comercio donde el cliente realizó la transacción -Esto no hace parte del control de la lógica de la API, es solo un contexto para el modelo de datos-).
- El checkpoint debe ser aplicado solo a guías que se encuentren registradas en base de datos, de lo contrario no se procesa.
- Se debe garantizar que todos los checkpoints (cambios de estado) sean aplicados y puedan visualizarse en el tracking (rastreo de paquetes).
- Se estima que el volumen diario sea de 300.000 guías/día ⇒ 366.000 unidades/día ⇒ ~1.210.00 checkpoints/día (estimado), con un crecimiento interanual del 30% en guías.
- Se estima que las horas pico son: 9:00–11:00 AM (guías aproximadas 120.000 en el checkpoint de recolección) y 16:00–21:00 (checkpoint de entregas, checkpoint de

descargue de recolección, checkpoint de despacho de aproximadamente 130.000 unidades).

**Alcance funcional de la solución esperada:**

- Registrar checkpoints inmutables por unidad.
- Consultar tracking por trackingId (histórico y último estado).
- Listar **guías** por estado.
- Estados **sugeridos** por cada checkpoint: CREATED, PICKED_UP, IN_TRANSIT, AT_FACILITY, OUT_FOR_DELIVERY, DELIVERED, EXCEPTION.

**Requerimientos No Funcionales:**

- Usar C# con .NET 8+
- Aplique algún diseño de arquitectura y explique en el README.md del repo cual aplicó y por qué.
- Implemente seguridad de la API (Algunas sugerencias)
- Implemente test unitarios al menos a un módulo de la API.

**Mínimos contratos esperados:**

- POST /api/v1/checkpoints → Registrar checkpoint.
- GET /api/v1/tracking/:trackingId → Obtener historial.
- GET /api/v1/shipments → Listar guías y sus unidad(es) por estado.

**Entregables Etapa 1:**

- Código fuente del API en repositorio público de GitHub.
- API desplegada y funcional.

**Bloque 1 - Etapa 2: Revisión y Refactor de Código**

**Objetivo:** Identificar al menos 15 errores principales y proponer soluciones que demuestren dominio de SOLID, Clean Code, patrones de diseño y arquitectura limpia.

El ejemplo a continuación contiene malas prácticas intencionales:

Blackbird

```
// app.ts
import Fastify from "fastify";
class CheckpointManager {
  checkpoints: any[] = [];
  createCheckpoint(unitId: string, status: string, timestamp: Date) {
    this.checkpoints.push({
      id: Math.random().toString(),
      unitId,
      status,
      timestamp: timestamp.toISOString(),
      history: [],
    });
    return this.checkpoints;
  }
  getHistory(unitId: string) {
    return this.checkpoints.filter((c) => c.unitId == unitId);
  }
}
class UnitStatusService {
  units: any[] = [];
  updateUnitStatus(unitId: string, newStatus: string) {
    let unit = this.units.find((u) => u.id == unitId);
    if (!unit) {
      unit = { id: unitId, status: newStatus, checkpoints: [] };
      this.units.push(unit);
    }
    unit.status = newStatus;
    unit.checkpoints.push({ status: newStatus, date: new Date().toString() });
    return unit;
  }
  getUnitsByStatus(status: string) {
    return this.units.filter((u) => u.status == status);
  }
}
class TrackingAPI {
  checkpointManager = new CheckpointManager();
  unitService = new UnitStatusService();
  registerRoutes(app: any) {
```

Blackbird

```
    app.post("/checkpoint", async (req: any, reply: any) => {
      const { unitId, status } = req.body;
      const cp = this.checkpointManager.createCheckpoint(
        unitId,
        status,
        new Date()
      );
      this.unitService.updateUnitStatus(unitId, status);
      reply.send(cp);
    });
    app.get("/history", async (req: any, reply: any) => {
      const { unitId } = req.query as any;
      reply.send(this.checkpointManager.getHistory(unitId));
    });
    app.get("/unitsByStatus", async (req: any, reply: any) => {
      const { status } = req.query as any;
      reply.send(this.unitService.getUnitsByStatus(status));
    });
  }
}
const app = Fastify();
const api = new TrackingAPI();
api.registerRoutes(app);
app.listen({ port: 3000 }, (err: any, address: string) => {
  if (err) {
    process.exit(1);
  }
  console.log(`Server running at ${address}`);
});
```

**Guía para el/la developer:**

- Señalar problemas concretos en el código (ejemplo: violación de SOLID, acoplamiento excesivo, ausencia de validación, malas prácticas de persistencia, etc.).
- Para cada problema, indicar:

Blackbird

    - Principio afectado (Clean Code, SOLID, Clean Architecture, diseño de APIs, seguridad, etc.).
    - Riesgo asociado (mantenibilidad, escalabilidad, seguridad, consistencia, etc.).
- Separar el código en capas claras (ej. controladores, servicios, repositorios, dominio).
- Definir contratos e interfaces que permitan independencia entre capas.
- Usar Inyección de Dependencias (DI).
- Manejar transacciones donde aplique.
- Implementar idempotencia en la creación de checkpoints.
- Validar datos de entrada (unitId, status, fechas).
- Manejar adecuadamente los errores y excepciones (respuestas claras de error HTTP).

**Entregable Etapa 2:**

- Código refactorizado (usar el stack tech de preferencia/libre).