COORDINADORA

Prueba Coordinador de Tecnología (CT)

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 y de arquitectura de software.

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

Objetivo

Construir una API para registrar checkpoints y consultar el tracking de paquetes.

Contexto y escala operativa

- Un checkpoint es un toque/escaneo de una unidad para mantener la trazabilidad y asegurar la carga durante todo el ciclo logístico. El ciclo logístico tiene una duración máxima de 3 días por guía y se realizan en promedio 10 checkpoints por unidad (promedio 3.3 checkpoints por día por unidad). Cada guía transporta en promedio 1.22 unidades.
- El checkpoint debe ser aplicado solo a unidades 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 (unidades 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).

COORDINADORA

Alcance Funcional (MVP)

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

Requerimientos No Funcionales

- Clean Architecture (interfaces → application → domain → infrastructure).
- Seguridad APIs.
- Tests unitarios y de integración.

Contratos de la API

- POST /api/v1/checkpoints → Registrar checkpoint.
- GET /api/v1/tracking/:trackingId → Obtener historial.
- GET /api/v1/shipments → Listar unidades por estado.

Entregables Etapa 1:

- Código fuente del API en repositorio de GitHub.
- API desplegada y funcional.
- Diagramas de arquitectura C4 y decisiones arquitectónicas.

COORDINADORA

Criterios de Evaluación Etapa 1

✅ Código limpio y estructurado aplicando principios SOLID y Clean Architecture.
✅ Arquitectura y separación de capas (Diagramas C4).
✅ Correctitud funcional.
✅ Funcionales (tests, logs, métricas, seguridad).
✅ Performance/extensibilidad.
✅ Bonus: contenedores y CI/CD

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:

```
// 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);
    }
}
```

COORDINADORA

```
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) {
        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();
```

COORDINADORA

```
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 candidato/a

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

COORDINADORA

Entregables Etapa 2:

- Código refactorizado.

Criterios de Evaluación Etapa 2

✅ Detección de issues.
✅ Arquitectura objetiva y toma de decisiones.
✅ Calidad del refactor aplicado y tests.
✅ Manejo de errores.