Challenge Backend - Notifications Library

Crear una librería de notificaciones en Java que sea agnostica a algun framework y extendible.

Tu objetivo es **diseñar una arquitectura** que unifique el envío de notificaciones a través de diferentes canales (Email, Slack, SMS, etc.).

En otras palabras: Crea una abstracción que permita enviar notificaciones sin importar el canal, haciendo que cambiar de proveedor (SendGrid, Mailgun, Twilio, etc.) sea transparente para quien usa tu librería.

El enfoque está en la arquitectura y el diseño, no en hacer conexiones HTTP reales. Puedes simular el envío real.

> ⚠ **IMPORTANTE:** Todo el código de ejemplo en este documento es **solo referencial**. Las decisiones de diseño, arquitectura y software son **100% tuyas**. Siéntete libre de implementar la solución como consideres mejor. Los ejemplos solo buscan ilustrar la funcionalidad esperada, no dictar la implementación.

> **RECORDATORIO:** Esta es una **librería, NO una aplicación**. Debe ser **agnóstica a frameworks** (Spring, Quarkus, etc.). No uses anotaciones como `@Component`, `@Service`, ni archivos de configuración externos (YAML, properties). Todo debe configurarse mediante código Java puro. Tu librería debe poder usarse en cualquier proyecto Java, con o sin framework.

> **SÍ puedes usar:** Librerías de utilidad como Lombok (`@Getter`, `@Builder`), Jackson/Gson (JSON), SLF4J (logging), Apache Commons, etc. Estas son herramientas, no frameworks que fuerzan arquitectura.

Lo importante: Que tu diseño sea claro, intuitivo y bien justificado.

¿Qué evaluaremos?

- **Diseño de arquitectura** - ¿La abstracción es clara y extensible?
- **Principios SOLID** - Especialmente Open/Closed, Dependency Inversion y Single Responsibility
- **Patrones de diseño** - Factory, Strategy, Builder, etc.
- **Extensibilidad** - ¿Es fácil agregar un nuevo canal?
- **Calidad de código** - Código limpio y buenas prácticas
- **Testing** - Tests unitarios bien diseñados
- **Documentación** - Cómo usar y extender la librería

Lo que debes implementar

Resumen de Funcionalidades

Tu librería de notificaciones debe incluir:

- **Interfaz Común** - Abstracción unificada para todos los canales
- **3 Canales Obligatorios** - Email, Push Notification, SMS
- **Configuración** - Configuración mediante clases Java (no archivos)
- **Manejo de Errores** - Try/catch y validación de resultados
- **Tests Unitarios** - Con mocks/simulaciones
- **Documentación** - README con guía de uso y extensión

Opcional:

- **Dockerfile** - Empaqueta la librería y **ejemplos para ejecución fácil**
- Notificaciones asíncronas con `CompletableFuture`
- Sistema de reintentos
- Validación de notificaciones: Email válido, teléfono válido, etc.
- Templates de mensajes
- Notificación de estado de la notificación: Pub/Sub pattern o algún sistema de notificar a canales

---

1. Interfaz Común de Notificación

Lo que debe lograr:

- Interfaz unificada que funcione para todos los canales
- Mismo código para enviar Email, SMS, Push, etc.
- Facilitar el cambio entre canales sin modificar el código cliente

Tu decisión: Define la interfaz y los métodos que mejor se adapten a tu diseño.

2. Múltiples Canales de Notificación

Requisitos:

- **Email** (obligatorio) - Para notificaciones por correo
- **Push Notification** (obligatorio) - Para notificaciones móviles
- **SMS** (obligatorio) - Para mensajes de texto
- **Slack** (opcional) - Para mensajes en workspace

 Tu decisión: La misma interfaz debe funcionar para todos los canales. Diseña cómo manejar las diferencias entre canales (ej: Email tiene subject, SMS no).

3. Configuración

Lo que debe permitir:

- Configurar credenciales de proveedores (API keys, tokens, etc.)
- Configuración 100% mediante código Java (no archivos YAML/properties)
- Soportar múltiples proveedores por canal (ej: SendGrid o Mailgun para Email)

 Tu decisión: Elige el patrón de configuración que prefieras (Builder, Factory, etc.). Asegura que sea fácil agregar nuevos proveedores.

4. Manejo de Errores

Lo que debe cubrir:

- Distinguir entre errores de validación y errores de envío
- Información clara sobre qué falló
- Fácil de usar con try-catch

 Tu decisión: Define tu estrategia de manejo de errores (excepciones, Result types, códigos de error, etc.).

5. Notificaciones Asíncronas (Opcional)

Lo que debe lograr:

- Envío no bloqueante de notificaciones
- Usar `CompletableFuture` para manejo asíncrono
- Permitir envío en lote

 Tu decisión: Implementa si quieres destacar. Define cómo manejar errores en contexto asíncrono.

Requisitos Técnicos

Obligatorios

- **Java 21 o superior**
- **Build tool**: Maven
- **3 canales obligatorios**: Email, Push Notification y SMS
- **Principios SOLID** aplicados correctamente
- **Arquitectura extensible** - Fácil agregar nuevos canales sin modificar código existente
- **Tests unitarios** - No necesitas hacer conexiones reales, usa simulaciones/mocks
- **Código limpio y fácil de entender**

Sobre el envío real de notificaciones

No necesitas hacer conexiones HTTP reales. Puedes:

- Simular el envío (logs, mensajes en consola)
- Usar mocks o stubs
- Retornar resultados simulados

El enfoque está en la arquitectura, no en la integración real con APIs externas.

> **Nota importante:** Aunque no implementes las conexiones reales, **sí debes revisar la documentación de los proveedores** (SendGrid, Twilio, Firebase Cloud Messaging, etc.) para entender cómo funcionan sus APIs. Tu diseño debe reflejar que entiendes qué datos necesita cada proveedor, qué formato esperan, y qué respuestas devuelven. Esto demuestra que tu arquitectura es realista y podría implementarse con proveedores reales.

Lo que esperamos recibir

1. Código

- Tu implementación de la librería
- 3 canales obligatorios: Email, Push Notification y SMS
- Tests unitarios bien diseñados
- Ejemplos de uso
- **Importante:** No necesitas integraciones reales, simula el envío

2. README.md

Escribe un README pensando en otro desarrollador:

- **Instalación**: ¿Cómo se usa? (Maven/Gradle)
- **Quick Start**: Un ejemplo simple
- **Configuración**: Cómo configurar cada canal y proveedor

- **Proveedores soportados**: Lista de integraciones
- **API Reference**: Clases y métodos principales
- **Seguridad**: Mejores prácticas para manejar credenciales

3. Dockerfile (Opcional pero valorado)

Si quieres facilitar que otros prueben tu librería, incluye un Dockerfile que empaquete todo:

¿Qué debería hacer tu Dockerfile?

- Empaquetar la librería compilada
- Incluir ejemplos de uso ejecutables
- Permitir ejecutar demos sin configurar Java localmente

```
 FROM eclipse-temurin:21-jdk-alpine
WORKDIR /app
# Copiar archivos del proyecto
COPY . .
# Compilar la librería (Maven)
RUN ./mvnw clean package -DskipTests
# Ejecutar clase de ejemplos
# Opción 1: Si creaste un Main con ejemplos
CMD ["java", "-cp", "target/notifications-lib-1.0.0.jar", "com.example.notifications.examples.NotificationExamples"]
# Opción 2: Si creaste un módulo de ejemplos separado
# CMD ["java", "-jar", "examples/target/notification-examples.jar"]
```

Para luego construir y ejecutar la imagen

¿Cómo te evaluaremos?

Nos fijaremos en:

- **Diseño de la abstracción** - ¿Es clara y reutilizable?
- **Calidad del código** - ¿Está limpio? ¿Usa buenos patrones?
- **Seguridad** - ¿Manejas credenciales de forma segura?
- **Testing** - ¿Funciona? ¿Testeaste con mocks?
- **Documentación** - ¿Otro dev puede configurarlo sin ayuda?
- **Visión del proyecto** - ¿Tienes claro el objetivo y la dirección?

Sobre la completitud

No hay librería 100% terminada. Siempre hay cosas por mejorar y funcionalidades por agregar.

Si no llegas a completar todo el reto, **no te preocupes**. También evaluaremos:

- Tu **visión del proyecto**: ¿Tienes claro hacia dónde va?
- Tu **capacidad de priorización**: ¿Qué decidiste implementar primero?
- Tu **claridad de pensamiento**: ¿Documentaste qué falta y por qué?

Lo importante es que tengas claro el objetivo y puedas comunicar tu proceso de pensamiento, aunque no hayas implementado todo.

No buscamos perfección, buscamos ver cómo piensas y cómo resuelves problemas.

Consejos

1. **Empieza con un canal**: Primero diseña Email, luego replica para los demás
2. **Abstracción primero**: Define bien las interfaces antes de implementar
3. **Piensa en extensibilidad**: ¿Cómo alguien agregaría un nuevo canal sin tocar código existente?
4. **Simula, no integres**: No pierdas tiempo con APIs reales, enfócate en el diseño
5. **SOLID es clave**: Open/Closed principle es especialmente importante aquí
6. **Documenta decisiones**: Explica por qué elegiste cierta arquitectura

Tiempo

Dedícale el tiempo que consideres necesario. En promedio, esto toma entre **3 días**.

Al no necesitar integraciones reales, puedes enfocarte en el diseño y arquitectura.

No hay prisa - preferimos calidad sobre velocidad.

Inspiración

Si quieres ver cómo otros lo hacen:

- Notify - Librería Go unificada
- Notifiers - Librería Python unificada
- SendGrid API - Ejemplo de API de Email
- Slack Webhooks - Ejemplo de API de Slack

Uso de IA

Siéntete libre de usar herramientas de IA (GitHub Copilot, ChatGPT, Claude, etc.) durante el desarrollo.

De hecho, **nos interesa saber cómo trabajas con IA**:

- ¿Qué modelo o herramienta utilizaste?
- ¿Cómo fue tu proceso de trabajo con la IA?
- ¿Qué prompts o estrategias usaste?
- ¿Qué decisiones tomaste tú vs qué sugirió la IA?
- ¿En qué te ayudó y en qué no?

Y si no lo utilizas, también está bien. Este punto lo veremos en la entrevista técnica si pasas a la siguiente etapa.

Entrega

Cuando termines:

1. Sube tu código a **GitHub** (o GitLab)
2. Asegúrate de que el README esté completo
3. Compártenos el link al repositorio

Siguiente Etapa

Si tu trabajo nos gusta y pasas a la siguiente etapa, **haremos una sesión de pair programming** donde:

- Revisaremos tu código juntos
- Discutiremos decisiones de diseño y alternativas
- Será una conversación técnica, no un examen

Esta sesión nos ayuda a ver cómo trabajas en equipo y cómo piensas en tiempo real.

¿Dudas?

Si tienes preguntas sobre el challenge:

- Pregunta lo que necesites
- No hay "preguntas tontas"
- Queremos ver cómo enfrentas ambigüedad y tomas decisiones

¡Mucha suerte!