Refactorización Propuesta

A continuación, se presentan las malas prácticas encontradas y la solución propuesta:

Aplicación del Principio de Responsabilidad Única (SRP):

- **Problema**: El controlador original tenía múltiples responsabilidades. El método analyze hacía demasiado: determinaba el tipo de red social, analizaba los mensajes, interactuaba con la base de datos y determinaba el nivel de riesgo, mejor dicho hacía todo en un solo método
- **Solución**: El controlador fue simplificado para delegar la lógica de negocio al servicio AnalyzeSocialMentionUseCase. El servicio se encarga de las diferentes responsabilidades de forma modular. Se crean las clases de repository encargadas de insertar la data en base de datos.

Nueva Estructura de Clases

- **Problema**: El código hace mucho uso de ifs, lo que hace complejo el entendimiento y mantenibilidad del código
- **Solución**: El uso de una fábrica (Factory) y el patrón Strategy elimina la necesidad de usar if para determinar el tipo de mención, haciendo el código más limpio y fácil de mantener.

1. **Clase Base SocialMention**: Esta clase abstracta sirve como punto común para las menciones sociales, pero ahora cada tipo de mención se manejará en subclases específicas.
2. **Subclases Específicas (FacebookMention, TwitterMention)**: Estas clases heredan de SocialMention y encapsulan la lógica específica para cada tipo de mención.
3. **Interfaz SocialAnalyzerService**: Define el método analyze que será implementado por diferentes analizadores de redes sociales.
4. **Implementaciones de SocialAnalyzerService (FacebookAnalyzerService, TwitterAnalyzerService)**: Implementan la lógica específica para analizar menciones en cada red social.
5. **SocialAnalyzerFactory**: Una fábrica que decide qué analizador utilizar basado en el tipo de mención.

Uso de DDD

- **Problema**: El código no hace uso de alguna arquitectura, todo se hace en un mismo método
- **Solución**: Usar una estructura modular en el proyecto separando los adaptadores (in/out), la capa aplicación que contiene los casos de uso, y la capa dominio que contiene las entidades, agregados, values objects. etc

Código sin pruebas

- **Problema**: El código propuesto no tiene pruebas, y por ende no se asegura que el código haga lo que se espera que haga, además agregar nuevas funcionalidades es un riesgo porque no existen pruebas que aseguren que los nuevos cambios rompan la lógica ya implementada
- **Solución**: Se agregan clases de prueba a los adaptadores y caso de uso.