Corporación Universitaria del Huila – CORHUILA

Asignatura: Arquitectura de Software

ANÁLISIS – FASE 2

Análisis de Anti-Patrones y Malas Prácticas

Docente:
Luis Ángel Vargas Narvaez

Integrantes:
Juan José Torrejano Rojas - jjtorrejano-2032b@corhuila.edu.co
Jhon Edinson Marín Tapias - jemarin-2032b@corhuila.edu.co
Karina Cantillo Plaza - kcantillo-2032b@corhuila.edu.co
Danay Mariana Pereira Ospina - dmpereira-2032b@corhuila.edu.co

Neiva, Huila

15 de febrero de 2026

Introducción

En la presente fase se realizó el análisis técnico del sistema “Espagueti de Encuestas” con el objetivo de identificar anti-patrones y malas prácticas presentes tanto en el backend desarrollado en Spring Boot como en el frontend implementado en Angular.

El análisis se centró en la revisión manual del código fuente, clasificando los hallazgos en las siguientes categorías: Seguridad, Arquitectura, Principios SOLID, Clean Code, Tipado y Rendimiento. Esta fase tiene un enfoque diagnóstico, cuyo propósito es evidenciar los problemas técnicos existentes sin realizar aún modificaciones al sistema.

Actividad 2.1 - Análisis del Backend (Java/Spring Boot)

Archivo analizado: EncuestaController.java

A continuación, se presenta la tabla consolidada de anti-patrones identificados en el backend, clasificados por categoría, indicando las líneas específicas del código y su respectiva justificación técnica e impacto en el sistema.

Categoría	Anti-Patrón	Líneas	Justificación / Impacto
Seguridad	SQL Injection (concatenación SQL)	41, 78, 101–105, 109	Las consultas se construyen concatenando entrada del usuario, lo que habilita inyección SQL y acceso no autorizado. Debe usarse SQL parametrizado (PreparedStatement/JdbcTemplate con ?).
Seguridad	Credenciales hardcodeadas	16–18	Credenciales en el código fuente exponen secretos y dificultan el manejo por ambientes. Usar variables de entorno/Config Server.
Arquitectura	Sin capas (Service, Repository)	Todo el archivo	El Controller realiza validaciones, lógica de negocio y acceso a datos, aumentando acoplamiento y complejidad.
SOLID	Violación SRP (Single Responsibility)	Controlador hace todo	Una sola clase asume múltiples responsabilidades, afectando mantenibilidad y testabilidad.
Clean Code	Código duplicado (validaciones)	33, 60, 75, 90, 96	Validaciones repetidas incrementan errores y el costo de mantenimiento. Extraer validadores reutilizables.
Manejo de Errores	printStackTrace() y return null	51–53, 66–68, 82–84, 112–114	Expone detalles internos y devuelve respuestas inconsistentes. Usar manejo centralizado de excepciones (ControllerAdvice).
Tipado	Uso de Map genérico sin DTOs	30, 46, 72, 87	Pierde contratos de tipos y validación. Definir DTOs para requests/responses.

Rendimiento	Conexión BD recreada por llamada	20–27 (método jdbc())	Crear DataSource por request degrada rendimiento. Usar DataSource con pooling gestionado por Spring.

Actividad 2.2 - Análisis del Frontend (Angular)

A continuación, se presenta la tabla consolidada de anti-patrones identificados en el frontend, organizados por categoría, especificando el archivo, las líneas de código y la justificación técnica de cada hallazgo.

Categoría	Anti-Patrón	Archivo	Líneas	Justificación / Impacto
Seguridad	URL API hardcodeada	crear.component.ts	18	La URL del backend está fija en el componente. Debe moverse a environment.ts para soportar ambientes (dev/prod).
Seguridad	URL API hardcodeada	encuesta.component.ts	18	Repetición de URL en múltiples componentes aumenta mantenimiento y riesgo de errores.
Seguridad	URL API hardcodeada	respuestas.component.ts	15	Configuración sensible en el código del componente en lugar de configuración centralizada.
Arquitectura	HttpClient directo (sin Service)	crear.component.ts	21	El componente consume la API directamente. Viola el patrón Service; dificulta pruebas y reutilización.
Arquitectura	HttpClient directo (sin Service)	encuesta.component.ts	22	Acopla la vista a la comunicación HTTP; debería existir un servicio (EncuestaService).
Arquitectura	HttpClient directo (sin Service)	respuestas.component.ts	18	Rompe separación de responsabilidades y duplica lógica de acceso a API.
Clean Code	Manipulación DOM directa (document.get ElementById)	home.component.ts	18	Angular promueve binding y @ViewChild; el DOM directo rompe el enfoque declarativo.
Clean Code	Manipulación DOM directa	crear.component.ts	29	Código frágil y difícil de testear al manipular el DOM directamente.

	(document.get ElementById)			
SOLID	Uso excesivo de 'any' (sin interfaces)	Varios componentes	Múltiples líneas	Se pierden contratos de tipos. Definir interfaces (Encuesta, Respuesta, ApiResponse).
Rendimiento	Polling manual con setInterval en lugar de RxJS	encuesta.component.ts	27–29	Genera consumo innecesario y posibles fugas. Usar RxJS (interval, switchMap, takeUntil).

Conclusiones de la Fase 2

En la Fase 2 se identificaron múltiples anti-patrones tanto en el backend como en el frontend del sistema. En el backend, los problemas más críticos se relacionan con seguridad, especialmente la posibilidad de inyección SQL y el uso de credenciales hardcodeadas. Además, se evidenció ausencia de separación por capas, violación del principio de Responsabilidad Única, duplicación de código, manejo inadecuado de errores y problemas de rendimiento por la recreación constante de la conexión a base de datos.

En el frontend se detectaron malas prácticas como URLs de la API hardcodeadas, uso directo de HttpClient sin un servicio intermedio, manipulación directa del DOM, uso excesivo del tipo *any* y polling manual con *setInterval*. Estas prácticas afectan la mantenibilidad, escalabilidad y calidad del código.

En general, aunque el sistema funciona a nivel funcional, presenta debilidades estructurales importantes que requieren refactorización para mejorar su seguridad, organización arquitectónica y rendimiento.