

Corporación Universitaria del Huila – CORHUILA

Asignatura: Arquitectura de Software

**PROPUESTA – FASE 3**  
**ADRs (Architecture Decision Records)**

Docente:  
Luis Ángel Vargas Narváez

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

Neiva, Huila  
23 de febrero de 2026

## ADR-001 · Adoptar arquitectura de microservicios

Estado	Accepted
Contexto	El monolito actual mezcla acceso a datos, lógica de negocio y presentación en EncuestaController.java. Las funcionalidades de creación y votación tienen cargas distintas, dificultando el escalado independiente.
Decisión	Descomponer el monolito en dos microservicios: survey-service (CRUD) y voting-service (votos), usando contextos delimitados.
Consecuencias	<p>Escalabilidad independiente según la demanda de cada servicio.</p> <p>Aislamiento de fallos (si falla el servicio de votos, se pueden seguir creando encuestas).</p> <p>Mayor complejidad operacional (requiere Service Discovery y Config Server).</p>

## ADR-002 · Usar Spring Data JPA en lugar de JDBC manual

Estado	Accepted
Contexto	En la Fase 2 se detectó que el código construye consultas SQL mediante concatenación de strings (líneas 41, 78, 101-105). Esto expone la aplicación a ataques de <b>Inyección SQL</b> y genera código difícil de mantener.
Decisión	Sustituir el acceso a datos manual por <b>Spring Data JPA</b> . Se utilizarán entidades anotadas con <code>@Entity</code> e interfaces que extiendan de <code>JpaRepository</code> .
Consecuencias	<p>Eliminación de riesgos de seguridad mediante el uso automático de <i>Prepared Statements</i>.</p> <p>Reducción drástica de código repetitivo (<i>Boilerplate</i>).</p> <p>Curva de aprendizaje inicial en el manejo de estados de entidades y <i>Lazy Loading</i>.</p>

### ADR-003 · Implementar Arquitectura en Capas (Layered Architecture)

<b>Estado</b>	Accepted
<b>Contexto</b>	<code>EncuestaController.java</code> viola el principio de Responsabilidad Única (SRP) al realizar todas las tareas del sistema. El análisis de la Fase 2 confirmó la ausencia de capas de Servicio y Repositorio.
<b>Decisión</b>	Organizar el backend en tres capas lógicas:  <ol style="list-style-type: none"><li>1. <b>API/Controller:</b> Manejo de peticiones HTTP.</li><li>2. <b>Service:</b> Implementación de reglas de negocio y validaciones.</li><li>3. <b>Repository:</b> Persistencia de datos.</li></ol>
<b>Consecuencias</b>	Código más organizado y fácil de testear de forma aislada.  Cumplimiento de estándares de la industria para aplicaciones Spring Boot.

### ADR-004 · Uso de DTOs y Eliminación de Mapas Genéricos

<b>Estado</b>	Accepted
<b>Contexto</b>	El uso de <code>Map&lt;String, Object&gt;</code> (líneas 30, 46, 72, 87) impide tener un contrato claro con el Frontend, dificulta la validación de datos y obliga a realizar conversiones de tipo manuales riesgosas.
<b>Decisión</b>	Adoptar el <b>Patrón DTO (Data Transfer Object)</b> . Se utilizarán clases POJO específicas para la entrada y salida de datos, desacoplando el modelo de base de datos de la interfaz externa.
<b>Consecuencias</b>	Tipado fuerte y mejor documentación del API.  Posibilidad de usar validaciones automáticas con Jakarta Validation.

## ADR-005 · Centralización de Lógica en Servicios de Angular

<b>Estado</b>	Accepted
<b>Contexto</b>	En el frontend, los componentes realizan llamadas directas a la API y tienen la URL escrita a mano ( <code>crear.component.ts:18</code> , <code>encuesta.component.ts:18</code> ). Esto viola la arquitectura sugerida por Angular.
<b>Decisión</b>	Crear un servicio inyectable ( <code>EncuestaService</code> ) que centralice todas las peticiones HTTP y maneje la configuración de la URL base desde el archivo <code>environment.ts</code> .
<b>Consecuencias</b>	Facilidad para cambiar la URL de la API en un solo lugar.  Reutilización de la lógica de comunicación en múltiples componentes.

## ADR-006 · Sustitución de Polling Manual por RxJS

<b>Estado</b>	Accepted
<b>Contexto</b>	El uso de <code>setInterval</code> en <code>encuesta.component.ts</code> (líneas 27-29) para actualizar resultados es ineficiente y propenso a fugas de memoria si no se gestiona el ciclo de vida del componente.
<b>Decisión</b>	Implementar el polling mediante operadores reactivos de <b>RxJS</b> ( <code>interval</code> , <code>switchMap</code> ). Se asegurará la limpieza de la suscripción mediante el uso de <code>takeUntil</code> en el evento <code>ngOnDestroy</code> .
<b>Consecuencias</b>	Gestión de memoria optimizada y flujo de datos asíncrono robusto.  Requiere conocimiento especializado en programación reactiva.

## ADR-007 · Manejo Global de Excepciones y Clean Code

Estado	Accepted
Contexto	El manejo actual de errores usa <code>printStackTrace()</code> y retorna <code>null</code> (líneas 51-53, 82-84), lo que oculta fallos y no informa adecuadamente al cliente. Además, existen nombres de variables no significativos como <code>r o e</code> .
Decisión	<ol style="list-style-type: none"><li>1. Implementar un <b>Global Exception Handler</b> con <code>@ControllerAdvice</code>.</li><li>2. Refactorizar el código siguiendo principios de <b>Clean Code</b> (nombres significativos, funciones pequeñas).</li></ol>
Consecuencias	Respuestas de API estandarizadas y profesionales.  Código mucho más legible y mantenible por otros desarrolladores.

## ADR-008 · Externalización de Configuración y Seguridad

Estado	Accepted
Contexto	Se identificó el uso de credenciales de base de datos expuestas en el código fuente (líneas 16-18), lo cual es una vulnerabilidad crítica de seguridad.
Decisión	Moverx todas las credenciales y configuraciones variables a variables de entorno y archivos <code>application.yml</code> , asegurando que no se suban secretos al sistema de control de versiones.
Consecuencias	Seguridad mejorada.  Flexibilidad para desplegar en diferentes entornos (Desarrollo, QA, Producción).

## ADR-009 · Implementación de API Gateway

<b>Estado</b>	Accepted
<b>Contexto</b>	Al dividir el sistema en microservicios, el frontend tendría que conocer múltiples URLs y gestionar problemas de CORS (Cross-Origin Resource Sharing) de forma individual.
<b>Decisión</b>	Implementar un <b>API Gateway</b> como punto de entrada único para todas las peticiones del frontend. Este componente se encargará del ruteo, la agregación de respuestas y la seguridad perimetral.
<b>Consecuencias</b>	<p>Simplifica la comunicación desde el cliente (Angular solo conoce una URL).</p> <p>Centralización de logs, métricas y autenticación.</p> <p>Se convierte en un posible cuello de botella si no se escala adecuadamente.</p>