Corporación Universitaria del Huila - CORHUILA

Arquitectura de Software

Taller - Fase 3

Propuestas de Rediseño Arquitectónico

Docente:

Luis Ángel Vargas Narváez

Integrantes:

Angie Valentina Flórez Vargas — Avflorez-2023a@corhuila.edu.co

Sebastián Puentes Gonzales — Spuentes-2022b@corhuila.edu.co

Sergio Alejandro Muñoz Cabrera — Samunoz-2023a@corhuila.edu.co

Miguel Ángel Rivera Lozano — marivera-2023ª@corhuila.edu.co

Neiva, Huila

17 de Febrero 2026

Propuestas
de
Rediseño Arquitectónico

3.1 - TABLA DE PATRONES DE DISEÑO

EN ESTA FASE SE PROPONE EL REDISEÑO ARQUITECTÓNICO DEL SISTEMA "ESPAGUETI DE ENCUESTAS", PASANDO DE UNA ARQUITECTURA MONOLÍTICA A UNA ARQUITECTURA BASADA EN MICROSERVICIOS. SE IDENTIFICARON ANTI-PATRONES PRESENTES EN EL CÓDIGO ORIGINAL Y SE PROPONEN PATRONES DE DISEÑO ALINEADOS CON BUENAS PRÁCTICAS DE ARQUITECTURA EMPRESARIAL, CLEAN CODE Y PRINCIPIOS SOLID.

Patrón	Anti-patrón detectado	Archivo	Líneas de código	Justificación	Ubicación en C4 propuesto
Repository Pattern (Spring Data JPA)	SQL concatenado (riesgo SQL Injection)	EncuestaController.java	• /crear: ~34–46 • /encuesta/{id}: ~36–51 • /votar: ~53–63 y ~64–71 • (Fase 2: 41, 78, 101–105, 109)	Prepared statements automáticos, menos boilerplate, seguridad y mantenibilidad	**survey-service** y **voting-service** (capa repositorio)
Arquitectura en Capas (Controller → Service → Repository)	El Controller hace todo / SRP roto	EncuestaController.java	Archivo completo	Separa responsabilidades (validación, negocio, acceso a datos), mejora pruebas y escalabilidad	Ambos microservicios del dominio

Propuestas
de
Rediseño Arquitectónico

Patrón	Anti-patrón detectado	Archivo	Líneas de código	Justificación	Ubicación en C4 propuesto
DTO Pattern + Bean Validation	Uso de Map genérico y validaciones duplicadas	EncuestaController.java	• crear(): ~23–46 • validación: ~26–32 • votar(): ~53–71	Contratos tipados, @NotBlank / @Size, elimina duplicación y errores	Capa API de **survey-service** y **voting-service**
Global Exception Handler (@RestControllerAdvice)	printStackTrace() + return null	EncuestaController.java	• catch: ~42–46 • catch: ~50–51 • catch: ~70–71	Estándar de errores, HTTP consistentes (400–500), mejora observabilidad	Cross-cutting en cada microservicio
Externalized Configuration	Credenciales hardcodeadas y DataSource manual	EncuestaController.java	• URL/USER/PASS: ~13–18 • jdbc(): ~19–27	Seguridad por ambiente, uso de Hikari, despliegue profesional	Gateway y servicios con configuración centralizada
Service Layer (Angular)	Uso directo de HttpClient con URLs hardcodeadas en componentes	crear.component.ts encuesta.component.ts respuesta.component.ts	• crear: url l.18 / post l.34 • encuesta: url l.18 / get l.38 / post l.54 • respuestas: url l.15 / get l.25	Reutilización, testabilidad y baseURL única (environment)	Angular SPA (cliente)
Reactive Polling (RxJS)	Uso imperativo de setInterval para polling	encuesta.component.ts	l.27–29 (intervalo) l.32–34 (clearInterval)	timer + switchMap + takeUntilDestroyed, evita leaks y mejora UX	Angular SPA (cliente)

Propuestas
de
Rediseño Arquitectónico

Patrón	Anti-patrón detectado	Archivo	Líneas de código	Justificación	Ubicación en C4 propuesto
Evitar manipular directamente el DOM	document.getElementById(...).innerHTML	crear.component.ts encuesta.component.ts home.component.ts	• crear: l.28–29 • encuesta: l.44–47 • home: l.17–18	Angular es declarativo: usar data binding, Renderer2 o ViewChild	Angular SPA (cliente)
HTTP Interceptor (Angular)	Manejo de errores HTTP repetido en componentes (duplicación de try/catch y mensajes inconsistentes)	crear.component.ts encuesta.component.ts respuesta.component.ts	• crear: l.34 (post) • encuesta: l.38 (get) / l.54 (post) • respuestas: l.25 (get)	Centraliza errores HTTP, headers (Authorization/ Content-Type), logging y mensajes. Reduce duplicación y asegura respuestas consistentes (mejor UX).	Angular SPA (cliente)
Environment Configuration (Angular)	Base URL del backend definida en componentes (acoplamiento a infraestructura / cambios manuales por entorno)	crear.component.ts encuesta.component.ts respuesta.component.ts	• crear: url l.18 • encuesta: url l.18 • respuestas: url l.15	BaseURL única en environment.ts (DEV/QA/PROD). Facilita despliegue, reduce errores y evita cambios repetidos en varios archivos.	Angular SPA (cliente)

3.2.1 - LISTA DE MEJORAS CLEAN CODE

Propuestas
de
Rediseño Arquitectónico

3.2.1 BACKEND (SPRING BOOT)

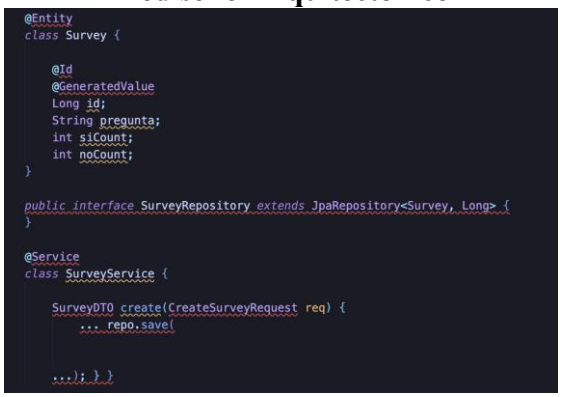
1. **Eliminar credenciales hardcodeadas**
   - **Archivo + línea:** EncuestaController.java ~13–18.
   - **Problema:** secretos y URL en código; jdbc() manual ignora el pool.
   - **Propuesta:** variables de entorno en application.yml y **eliminar jdbc() artesanal**.
   - **Snippet** (application.yml):

```yaml
spring:
  datasource:
    url: ${DB_URL}
    username: ${DB_USER}
    password: ${DB_PASS}
```

   - **Cambios de código:** inyectar JdbcTemplate ó pasar a JPA (recomendado por la guía) y quitar DriverManagerDataSource.
2. **Sustituir SQL concatenado por JPA (Repository Pattern)**

- **Archivo + línea:** EncuestaController.java ~34–46 (insert), ~36–51 select, ~53–63 update, ~64–71 select.
- **Problema:** exposición a **SQL Injection** + duplicación de lectura de entidad.
- **Propuesta:** @Entity Survey, SurveyRepository extends JpaRepository, SurveyService.
- **Snippet** (ilustrativo):

Propuestas
de
Rediseño Arquitectónico



- **Justificación:** Seguridad, mantenibilidad, SPR.(La guía recomienda JPA)

3. DTO + Bean Validation (reemplazar Map)

- **Archivo + línea:** EncuestaController.java método crear(@RequestBody Map) ~23–46; votar(@RequestBody Map) ~53–71; validación manual ~26–32.
- **Propuesta:**

- **Efecto:** DRY, contrato claro, errores 400 automáticos por @Valid.

4. Manejo de errores consistente (no return nul)

- **Archivo + línea:** EncuestaController.java jdbc() ~19–27.
- **Propuesta:** Inyectar un JdbcTemplate o EntityManager administrado por Spring.

3.2.2 Frontend (Angular) — Lista de Mejoras Clean Code

Propuestas
de
Rediseño Arquitectónico

1) Service Layer (Angular) — Evitar HttpClient directo en componentes

Archivo + línea:

- crear.component.ts → url l.18 / post l.34
- encuesta.component.ts → url l.18 / get l.38 / post l.54
- respuesta.component.ts → url l.15 / get l.25

Problema:

Los componentes consumen directamente HttpClient y definen URLs; esto duplica lógica, dificulta pruebas y acopla UI a infraestructura.

Propuesta:

Crear un servicio único (survey.service.ts) que concentre endpoints y retorne Observables tipados. Mantener baseURL en environment.ts.

Snippet (survey.service.ts):

```
@Injectable({ providedIn: 'root' })
export class SurveyService {

  private baseUrl = environment.apiUrl;

  constructor(private http: HttpClient) {}

  createSurvey(data: any) {
    return this.http.post(`${this.baseUrl}/crear`, data);
  }

  getSurvey(id: number) {
    return this.http.get(`${this.baseUrl}/encuesta/${id}`);
  }

  vote(data: any) {
    return this.http.post(`${this.baseUrl}/votar`, data);
  }
}
```

Efecto:

Reutilización, testabilidad, reducción de duplicación y baseURL única por entorno.

2) Reactive Polling (RxJS) — Reemplazar setInterval

Archivo + línea:

Propuestas
de
Rediseño Arquitectónico

- encuesta.component.ts → l.27–29 (intervalo) / l.32–34 (clearInterval)

Problema:

El uso manual de setInterval puede generar memory leaks y dificulta controlar el ciclo de vida del componente.

Propuesta:

Usar flujo reactivo con timer + switchMap y cancelar automáticamente con takeUntilDestroyed.

Snippet (encuesta.component.ts):

```
import { timer } from 'rxjs';
import { switchMap } from 'rxjs/operators';
import { takeUntilDestroyed } from '@angular/core/rxjs-interop';

timer(0, 3000).pipe(
  switchMap(() => this.surveyService.getSurvey(this.id)),
  takeUntilDestroyed(this.destroyRef)
).subscribe(data => this.survey = data);
```

Efecto:

Flujo declarativo, evita memory leaks y mejora UX.

3) Evitar manipular directamente el DOM

Archivo + línea:

- crear.component.ts → l.28–29
- encuesta.component.ts → l.44–47
- home.component.ts → l.17–18

Problema:

document.getElementById(...).innerHTML rompe el enfoque declarativo de Angular, dificulta mantenimiento y puede generar inconsistencias con el change detection.

Propuesta:

Usar **data binding** ({{ }}), *ngIf y [(ngModel)] (o Reactive Forms).

Snippet (template):

```
<p *ngIf="mensaje">{{ mensaje }}</p>
```

Propuestas
de
Rediseño Arquitectónico

Snippet (component):

```
this.mensaje = 'Encuesta creada correctamente';
```

Efecto:

Código más limpio, alineado con Angular, menos errores y mejor mantenibilidad.

4) HTTP Interceptor — Manejo centralizado de errores

Archivo + línea:

- crear.component.ts → l.34 (post)
- encuesta.component.ts → l.38 (get) / l.54 (post)
- respuesta.component.ts → l.25 (get)

Problema:

Cada componente maneja errores HTTP de forma diferente (duplicación) y no existe un estándar para mensajes o respuestas.

Propuesta:

Crear un HttpInterceptor para capturar errores, estandarizar manejo y opcionalmente registrar logs.

Snippet (error.interceptor.ts):

```
@Injectable()
export class ErrorInterceptor implements HttpInterceptor {

  intercept(req: HttpRequest<any>, next: HttpHandler) {
    return next.handle(req).pipe(
      catchError((error: HttpErrorResponse) => {
        const message = error.error?.message || 'Error de comunicación';
        return throwError(() => new Error(message));
      })
    );
  }
}
```

Efecto:

Propuestas
de
Rediseño Arquitectónico

Consistencia en errores, menos duplicación y mejor experiencia de usuario

5) Environment Configuration — Eliminar URLs hardcodeadas

Archivo + línea:

- crear.component.ts → url l.18
- encuesta.component.ts → url l.18
- respuesta.component.ts → url l.15

Problema:

La URL del backend está definida en componentes; cambiar de entorno requiere modificar varios archivos.

Propuesta:

Definir apiUrl en environment.ts y consumirla desde servicios.

Snippet (environment.ts):

```
export const environment = {
  production: false,
  apiUrl: 'http://localhost:8080'
};
```

Efecto:

Configuración por entorno (DEV/QA/PROD), despliegue más limpio y menos errores.