REPORTE DE ANÁLISIS - PROYECTO SPRING BOOT

Sistema de Gestión de Clientes

****Fecha de Análisis:**** 27 de Julio, 2025

****Analista:**** Jose Osvaldo Gonzalez

****Proyecto:**** restDemo-main

📋 **RESUMEN EJECUTIVO**

Este proyecto es una aplicación web completa desarrollada por Miguel instructor de Xideral en la primera semana, se realizó con ****Spring Boot**** que implementa un sistema CRUD (Create, Read, Update, Delete) para la gestión de clientes. La aplicación combina un backend REST API robusto con una interfaz frontend.

Este reporte no vera a detalle el front-end a petición de el instructor si no que será un resumen y características del back en que para eso fue creado este curso.

Características Principales:

- API REST completa para gestión de clientes
- Frontend responsivo con interfaz
- Integración con base de datos MySQL
- Arquitectura en capas bien definida
- Validación y manejo de errores

ARQUITECTURA DEL PROYECTO

ANÁLISIS TÉCNICO DETALLADO

1. CONFIGURACIÓN DEL PROYECTO

Maven Dependencies (pom.xml)

Spring Boot Parent: 3.5.4

Java Version: 17

Dependencias principales:

- `spring-boot-starter-web` - Framework web y REST
- `spring-boot-starter-data-jpa` - Persistencia de datos
- `mysql-connector-j` - Conector MySQL
- `spring-boot-starter-test` - Testing framework

Configuración de Base de Datos (application.properties)

- **Base de datos:** MySQL (localhost:3306/academy_db)
- **Usuario:** root
- **JPA:** Hibernate con MySQL Dialect

2. CAPA DE ENTIDADES

Entidad Cliente (`Cliente.java`)

```java
@Entity
@Table(name = "Cliente")
public class Cliente {
   @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;

   @Column(name = "nombre", nullable = false, length = 100)
```

```
    private String nombre;

    @Column(name = "email", nullable = false, length = 150, unique = true)
    private String email;

    @Column(name = "telefono", length = 20)
    private String telefono;

    @Column(name = "fecha_registro", columnDefinition = "TIMESTAMP DEFAULT CURRENT_TIMESTAMP")
    private LocalDateTime fechaRegistro;
}
```

3. CAPA DE REPOSITORIO

****ClienteRepository** (`ClienteRepository.java`)**

Extiende `JpaRepository<Cliente, Long>` proporcionando:

****Métodos automáticos de JPA:****

- `findAll()`, `findById()`, `save()`, `deleteById()`

****Métodos personalizados:****

- `findByEmail(String email)` - Buscar por email único
- `findByNombreContainingIgnoreCase(String nombre)` - Búsqueda parcial por nombre
- `findByTelefono(String telefono)` - Buscar por teléfono (usando @Query)
- `findAllOrderByFechaRegistroDesc()` - Ordenar por fecha de registro
- `existsByEmail(String email)` - Verificar existencia de email

4. CAPA DE SERVICIOS

****Interface IClienteService** (`IClienteService.java`)**

Define el contrato de servicios con 7 métodos principales.

****Implementación ClienteServiceImpl** (`ClienteServiceImpl.java`)**

****Funcionalidades implementadas:****

****CRUD completo**** con validaciones

****Validación de email único**** en creación y actualización

****Manejo de errores**** con excepciones descriptivas

****Búsquedas especializadas**** por diferentes campos

****Timestamp automático**** en creación de registros

****Validaciones implementadas:****

- Email único al crear/actualizar

- Existencia de cliente antes de actualizar/eliminar

- Asignación automática de fecha de registro

5. CAPA DE CONTROLADORES

**ClienteController (`ClienteController.java`)**

API REST completa con 8 endpoints:

| Método | Endpoint | Descripción |

| GET | `/api/clientes` | Obtener todos los clientes |

| GET | `/api/clientes/{id}` | Obtener cliente por ID |

| POST | `/api/clientes` | Crear nuevo cliente |

| PUT | `/api/clientes/{id}` | Actualizar cliente |

| DELETE | `/api/clientes/{id}` | Eliminar cliente |

| GET | `/api/clientes/email/{email}` | Buscar por email |

| GET | `/api/clientes/nombre/{nombre}` | Buscar por nombre |

| GET | `/api/clientes/telefono/{telefono}` | Buscar por teléfono |

| GET | `/api/clientes/ordenados` | Obtener ordenados por fecha |

CONFIGURACIÓN

Application.properties esta toda la configuración para enlace

6. FRONTEND

No se vera esto a petición del instructor ya que solo estará enfocado al back-end

Backend:

- API REST completa y bien documentada
- Validaciones robustas de datos
- Manejo apropiado de errores HTTP
- Queries JPA optimizadas
- Transacciones implícitas bien manejadas

Este proyecto es un ****excelente ejemplo didáctico**** de una aplicación Spring Boot completa. Demuestra comprensión sólida de creación de un CRUD. Para uso académico y aprendizaje, este proyecto es bueno para comprensión. Tomando en cuenta que se realizo con IA