Documento de Arquitectura del Sistema

Sistema de Gestión de Tickets

1. Introducción

1.1 Propósito del Documento

Este documento define la arquitectura técnica del **Sistema de Gestión de Tickets**, estableciendo los patrones de diseño, estructura de capas y decisiones técnicas que guiarán el desarrollo del proyecto.

---

2. Patrón de Arquitectura: Arquitectura en Capas

Se implementará una **Arquitectura en Capas (Layered Architecture)** que divide la aplicación en niveles con responsabilidades específicas y bien definidas.

2.1 Diagrama de Arquitectura

CAPA DE PRESENTACIÓN
API REST - Endpoints HTTP
• Recibe peticiones del cliente
• Valida formato de entrada
• Retorna respuestas estructuradas

↓

CAPA DE SEGURIDAD
Autenticación y Autorización
• Valida tokens JWT

• Controla acceso por roles
• Filtra peticiones no autorizadas

↓

CAPA DE DOMINIO
Lógica de Negocio
• Reglas de negocio del sistema
• Validaciones de estados
• Orquestación de operaciones

↓

CAPA DE PERSISTENCIA
Acceso a Datos
• Interacción con base de datos
• Consultas y transacciones
• Mapeo objeto-relacional

↓

BASE DE DATOS
Sistema Relacional

---

3. Descripción de las Capas

3.1 Capa de Presentación

Responsabilidad: Punto de entrada para las peticiones externas.

Funciones:

- Exponer endpoints REST
- Validar formato de datos de entrada
- Transformar respuestas a formato JSON
- Documentar API con OpenAPI/Swagger

3.2 Capa de Seguridad

Responsabilidad: Proteger el sistema y controlar el acceso.

Funciones:

- Autenticar usuarios mediante JWT
- Autorizar acceso según roles (USER, AGENT, ADMIN)
- Filtrar peticiones antes de procesarlas
- Manejar errores de autenticación

3.3 Capa de Dominio

Responsabilidad: Implementar la lógica de negocio del sistema.

Funciones:

- Aplicar reglas de negocio
- Validar estados y transiciones
- Orquestar operaciones entre entidades
- Generar notificaciones automáticas

Reglas de Negocio Principales:

1. Solo tickets en estado "Abierto" pueden ser asignados
2. Solo empleados con rol "Agente" pueden recibir asignaciones
3. Al asignar un ticket, cambia automáticamente a "En Progreso"
4. No se pueden reasignar tickets finalizados
5. Todo cambio de estado se registra para auditoría

3.4 Capa de Persistencia

Responsabilidad: Gestionar el almacenamiento y recuperación de datos.

Funciones:

- Realizar operaciones CRUD

- Ejecutar consultas personalizadas
- Manejar transacciones
- Convertir entre objetos de dominio y entidades de BD

Características:

- Uso de ORM (JPA/Hibernate)
- Repositorios para acceso a datos
- Mappers para conversión de objetos
- Consultas con JPQL

3.5 Capa de Configuración

Responsabilidad: Proporcionar configuraciones globales, configurar propiedades de aplicación.

---

4. Comunicación entre Capas

4.1 Flujo de Datos

Cliente HTTP

↓

[Presentación] - Recibe petición

↓

[Seguridad] - Valida autenticación/autorización

↓

[Dominio] - Aplica lógica de negocio

↓

[Persistencia] - Accede a base de datos

↓

[Base de Datos] - Almacena/recupera datos

↓

[Persistencia] - Convierte a objetos

↓

[Dominio] - Procesa resultado

↓

[Presentación] - Formatea respuesta

↓

Cliente HTTP

4.2 Principios de Comunicación

1. **Unidireccional:** Las capas solo pueden comunicarse con la capa inmediatamente inferior
2. **Desacoplamiento:** Cada capa es independiente y reemplazable
3. **Abstracción:** Las capas superiores no conocen detalles de implementación de las inferiores
4. **Transaccionalidad:** Las transacciones se manejan en la capa de persistencia

---

5. Patrones de Diseño Implementados

5.1 Layered Architecture (Arquitectura en Capas)

Propósito: Organizar el código en capas con responsabilidades separadas.

5.2 DAO Pattern (Data Access Object)

Propósito: Abstraer y encapsular el acceso a la fuente de datos.

Implementación:

- DAOs gestionan operaciones de persistencia
- Separan lógica de negocio del acceso a datos
- Facilitan el cambio de tecnología de persistencia

5.3 DTO Pattern (Data Transfer Object)

Propósito: Transferir datos entre capas sin exponer entidades internas.

Tipos de DTOs:

- **Request DTOs:** Datos de entrada del cliente
- **Response DTOs:** Datos de salida al cliente

5.4 Service Layer Pattern

Propósito: Encapsular la lógica de negocio en servicios reutilizables.

Características:

- Una interfaz de servicio por entidad de dominio
- Servicios orquestan operaciones complejas
- Transacciones manejadas declarativamente

5.5 Repository Pattern

Propósito: Abstraer la lógica de acceso a datos.

Implementación:

- Interfaces que definen operaciones de datos
- Implementación automática mediante Spring Data
- Queries personalizadas cuando es necesario

6. Justificación de la Arquitectura y Patrones para el Sistema de Tickets

6.1 ¿Por qué Arquitectura en Capas para este Sistema?

Se eligió la **Arquitectura en Capas** específicamente para el sistema de gestión de tickets porque:

- **Flujos complejos de asignación:** La asignación de tickets involucra validaciones de roles, cambios de estado, y notificaciones automáticas. La separación en capas permite manejar esta complejidad de forma ordenada.
- **Múltiples roles de usuario:** El sistema maneja USER, AGENT y ADMIN con permisos diferentes. La capa de seguridad centraliza este control sin contaminar la lógica de negocio.
- **Auditoría obligatoria:** Todo cambio de estado debe registrarse. La capa de dominio orquesta la creación del ticket record automáticamente, mientras la capa de persistencia lo almacena de forma transaccional.
- **Notificaciones automáticas:** Cuando un ticket se asigna o reasigna, múltiples notificaciones deben crearse. La capa de dominio coordina estas operaciones sin que la capa de presentación conozca estos detalles.
- **Crecimiento futuro:** El sistema puede expandirse con nuevas funcionalidades (comentarios, archivos adjuntos, SLA) agregándolas en las capas correspondientes sin reestructurar todo.

6.2 ¿Por qué estos Patrones para Gestión de Tickets?

DAO Pattern:

- Los tickets y asignaciones requieren consultas complejas (buscar por estado, por agente, por rango de fechas).
- El DAO centraliza estas consultas JPQL personalizadas sin mezclarlas con la lógica de negocio.

DTO Pattern:

- Al listar tickets de un agente, no necesitamos cargar todos los comentarios o el historial completo.
- Los DTOs permiten exponer solo la información relevante en cada contexto (lista vs detalle).
- Evita exponer contraseñas de empleados al retornar información de tickets.

Service Layer:

- La asignación de un ticket implica: validar rol, validar estado, cambiar estado, crear notificaciones y registrar el cambio.
- El servicio orquesta esta secuencia de operaciones en una transacción, garantizando consistencia.

Repository Pattern:

- Necesitamos buscar tickets por múltiples criterios (empleado, estado, prioridad, categoría).
- Spring Data JPA genera automáticamente métodos como findByEmployeeId() y findByState() sin escribir código SQL.