Plataforma de Streaming Musical Cliente-Servidor (Parte III: Spring Boot + MongoDB)

Resumen

En la Parte III se implementa el sistema completo utilizando **Java Spring Boot** con **MongoDB NoSQL**. Se mantiene toda la funcionalidad de las Partes I y II (usuarios, licencias, playlists, historial, logs, estadísticas, control parental, recomendaciones) pero con arquitectura web REST moderna, persistencia en BBDD y aplicación web como cliente.

Arquitectura Spring Boot + MongoDB

El sistema se estructura en tres capas siguiendo patrones Spring:

- **Capa Presentación**: Controladores REST @RestController
- **Capa Servicio**: @Service con lógica de negocio
- **Capa Persistencia**: @Repository con MongoRepository

Tecnologías del Sistema

Backend

- **Spring Boot 3.x**: Framework web + inyección de dependencias
- **Spring Data MongoDB**: Repositorios automáticos
- **Spring Security**: Autenticación JWT
- **Spring Validation**: Validación datos entrada

Base de Datos

- **MongoDB**: Colecciones para canciones, usuarios, licencias, logs

Cliente

- Aplicación web HTML/JavaScript consumiendo APIs REST

Clases y Anotaciones Spring

Entidad Cancion (MongoDB)

```
@Document(collection = "canciones")
public class Cancion {
    @Id private String id;
    private String titulo;
    private String artista;
    private int duracion;
    private int reproducciones;
    private boolean contenidoExplicito;
}
```

Repositorio MongoDB

```
@Repository
public interface CancionRepository
    extends MongoRepository<Cancion, String> {
    List<Cancion> findTop10ByOrderByReproduccionesDesc();
}
```

Servicio Canciones

```
@Service
public class CancionService {
    @Autowired private CancionRepository repo;

    public Cancion solicitarCancion(String id) { ... }
    public void incrementarReproducciones(String id) { ... }
}
```

Controlador REST

```
@RestController
@RequestMapping("/api/canciones")
public class CancionController {
    @Autowired private CancionService service;

    @GetMapping("/{id}")
    public ResponseEntity<Cancion> getCancion(@PathVariable String id) { ... }
}
```

Pasos de Implementación

Paso 1: Configuración Spring Boot

1. Crear proyecto Spring Boot con dependencias:

- `spring-boot-starter-web`
- `spring-boot-starter-data-mongodb`
- `spring-boot-starter-security`

2. Configurar conexión MongoDB en `application.properties`:

```
spring.data.mongodb.uri=mongodb://localhost:27017/streaming
```

Paso 2: Entidades MongoDB

1. `Cancion @Document`: título, artista, duración, reproducciones
2. `Usuario @Document`: nombre, contraseña, rol (ADMIN/STANDARD)
3. `Licencia @Document`: usuarioId, fechaInicio, fechaFin
4. `Log @Document`: tipo (ACCESO/REPRODUCCION/ERROR), timestamp

Paso 3: Repositorios

- `CancionRepository`: `findTop10ByOrderByReproduccionesDesc()`
- `UsuarioRepository`: `findByNombreAndContraseña()`
- `LicenciaRepository`: `findByUsuarioIdAndActivaTrue()`

Paso 4: Servicios de Negocio

- `CancionService`: `solicitarCancion(), validarLicencia()`
- `UsuarioService`: `login(), generarTokenJWT()`
- `LogService`: `registrarEvento()`

Paso 5: Controladores REST

```
POST /api/usuarios/login
GET  /api/canciones
GET  /api/canciones/{id}/reproducir
GET  /api/estadisticas
POST /api/playlists
```

Paso 6: Cliente Web

- HTML + JavaScript + Fetch API
- Autenticación JWT en headers
- Reproductor HTML5 <audio>

Funcionalidades Implementadas

Sistema de Usuarios

- Login REST con JWT
- Roles ADMIN/STANDARD
- Validación licencias activas

Reproducción

- Streaming HTTP Range requests
- Incremento contador reproducciones transaccional
- Control parental contenido explícito

Playlists e Historial

- Colecciones MongoDB separadas
- Historial circular (últimas 50 canciones)

Estadísticas Globales

- Endpoint `/api/estadisticas` con:
  - canción más reproducida
  - usuarios activos
  - tiempo total reproducido

Logs y Auditoría

- Colección MongoDB `logs`
- Eventos: login, reproducción, errores

Sistema Recomendaciones

- Basado en historial usuario + popularidad global
- Endpoint `/api/recomendaciones`

Seguridad Spring Security

Configuración JWT

- Filtro `JwtAuthenticationFilter`
- Validación token en cada request protegido
- Roles ADMIN/STANDARD en claims JWT

Control de Acceso

- `@PreAuthorize("hasRole('ADMIN')")` administradores
- `@PreAuthorize("hasRole('STANDARD')")` usuarios normales

Resultado final:

La clase `Application` debe:

1. Iniciar servidor Spring Boot puerto 8080
2. Cargar datos iniciales (canciones, usuarios admin)
3. Cliente web:
    - Login usuario estándar o admin
    - Consultar catálogo
    - Reproducir canciones
    - Crear playlist que pueda añadir o eliminar canciones
    - Ver recomendaciones
    - Consultar estadísticas
4. Verificar logs sólo para administradores