HISTORIA DE USUARIO

Sigue la siguiente estructura y agrega tareas de acuerdo con el número de semanas de cada módulo.

Nombre de la HU:	**Gestión Estándar de Errores y Seguridad JWT para Administración de Eventos y Venues**
Objetivo de la HU	• Diseñar un **esquema de manejo de errores estandarizado y mantenible**, compatible con clientes REST, y **habilitar seguridad robusta** mediante autenticación **stateless con JWT** y control de acceso por rol. El propósito es fortalecer la confiabilidad, trazabilidad y protección del sistema dentro de la arquitectura hexagonal.

TASK 1

Validaciones Avanzadas y Esquema de Errores Uniforme

Descripción de la Tarea:

1. Implementar **Bean Validation avanzada**:
   - Validaciones cruzadas entre atributos (por ejemplo, que `fechaInicio` sea menor que `fechaFin` en `EventRequest`).
   - Uso de **grupos de validación** (`Create`, `Update`) para diferenciar escenarios.
   - Definición de mensajes personalizados en `messages.properties`.

	2. Crear un **manejador global de errores** con `@ControllerAdvice` para capturar: • `MethodArgumentNotValidException`, `EntityNotFoundException`, `DataIntegrityViolationException`, entre otras. 3. Estandarizar la respuesta de error usando **ProblemDetail (RFC 7807)** con: • Campos `type`, `title`, `status`, `detail`, `instance`. • Inclusión de `timestamp` y `traceId` en cada respuesta. 4. Garantizar que los controladores REST de *Event* y *Venue* respondan bajo el mismo formato uniforme de error.
TASK 2 K	**Observabilidad y Logging Estructurado** Descripción de la Tarea: 1. Implementar **logging estructurado** con **SLF4J o Logback**, registrando: • Nivel (`INFO`, `WARN`, `ERROR`), endpoint afectado, usuario, tipo de error y timestamp. 2. Configurar un patrón de logs uniforme y legible desde la consola o archivo externo. 3. Asociar los `traceId` generados en las respuestas a los logs para correlacionar errores. 4. Asegurar que todos los eventos relevantes (autenticación fallida, validación, error 500) se registren con detalle, mejorando la observabilidad del sistema.
TASK 3 K	**Seguridad JWT y Control de Acceso por Rol** Descripción de la Tarea:

1. Configurar seguridad moderna usando **SecurityFilterChain** (sin `WebSecurityConfigurerAdapter`).
2. Implementar endpoints de **autenticación y registro**:
    - `/auth/register`: creación de usuario con contraseña cifrada mediante `PasswordEncoder`.
    - `/auth/login`: generación de **JWT** firmado con clave secreta y tiempo de expiración configurado.
3. Crear un **filtro JWT** para validar tokens y autenticar usuarios en cada petición.
4. Definir control de acceso:
    - Rutas protegidas con anotaciones como `@PreAuthorize("hasRole('ADMIN')")`.
    - Endpoints públicos (autenticación/registro) y privados (gestión de eventos y venues).
5. Configurar **CORS** y políticas de seguridad REST adecuadas.
6. Verificar que el sistema funcione en modo **stateless** (sin sesiones de servidor).

Criterios de Aceptación

- Respuestas de error estandarizadas bajo el formato **ProblemDetail** (RFC 7807).
- Inclusión de `timestamp` y `traceId` en cada error.
- Validaciones cruzadas y mensajes descriptivos activos.
- Logs estructurados que registran errores y trazas con contexto.
- Autenticación funcional basada en **JWT**.
- Roles y permisos aplicados correctamente con `@PreAuthorize`.
- Contraseñas cifradas y configuración **stateless** en toda la aplicación.

Story Points: 10

Cierre de la actividad

El sistema de **administración de eventos y venues** debe ofrecer un manejo de errores uniforme, trazable y entendible para clientes REST.
Además, debe incorporar un esquema de **seguridad sólida basada en JWT**, con control de acceso por rol y registro estructurado de errores.
La entrega debe incluir código funcional, documentación breve del formato de errores, estructura de roles y ejemplos de autenticación protegida.