26-11-2025

Compendium

Portal de Guías de Videojuegos

Asignatura: Desarrollo Web 2
Profesor: Vicente Zapata
Estudiante: Frank Bustamante

ÍNDICE

1. INTRODUCCIÓN

Objetivo del Proyecto

Desarrollar un sistema web completo utilizando el patrón MVC (Modelo-Vista-Controlador) con Spring Boot que permita gestionar una comunidad de guías de videojuegos, implementando operaciones CRUD funcionales con persistencia en base de datos H2.

Evolución desde la Evaluación 1

Aspecto	Evaluación 1	Evaluación 2
Arquitectura	Vistas HTML estáticas	Arquitectura MVC completa
Persistencia	Sin base de datos	H2 con JPA/Hibernate
Entidades	Datos simulados	9 entidades con relaciones
Operaciones	Solo visualización	CRUD completo funcional
Seguridad	Sin autenticación	Spring Security + roles

| Validaciones | Solo HTML5 | Bean Validation + lógica negocio |

| Gestión Archivos | No implementado | Upload imágenes/documentos |

Problemas Resueltos (documentados en `NOTAS_PARTE2.md`):

- Configuración de H2 Console (CSRF, frameOptions)

- Binding de MultipartFile en formularios

- Actualización de colecciones ManyToMany (clear + addAll)

- Query methods derivados post-refactor

2. DESARROLLO (Documentación del Código)

2.1 Modelo de Base de Datos

Entidades y Relaciones

El sistema implementa 9 entidades JPA con las siguientes relaciones:

Relaciones Many-to-Many (con tablas join):

- `Usuario` ↔ `Role` (usuarios_roles)

- `Juego` ↔ `Categoria` (juego_categorias)

- `Juego` ↔ `Plataforma` (juego_plataformas)

Relaciones Many-to-One (con FK directas):

- `Guia` → `Juego` (juego_id)

- `Guia` → `Usuario` (autor_id)

- `Archivo` → `Guia` (guia_id, CASCADE DELETE)

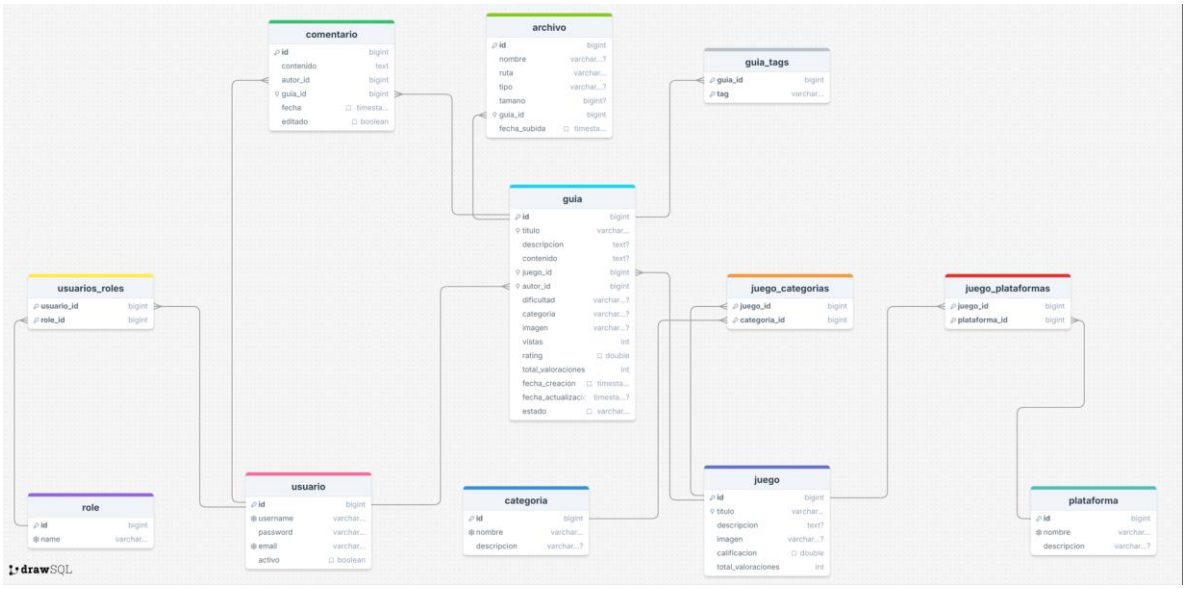
- `Comentario` → `Guia` (guia_id, CASCADE DELETE)

- `Comentario` → `Usuario` (autor_id)

Relación ElementCollection:

- `Guia` → `Tags` (guia_tags: guia_id + tag)

Diagrama MER



H2 Console — Verificación de Base de Datos

Acceso:

URL: http://localhost:8080/h2-console

JDBC URL: jdbc:h2:file:./data/compendium

User: sa

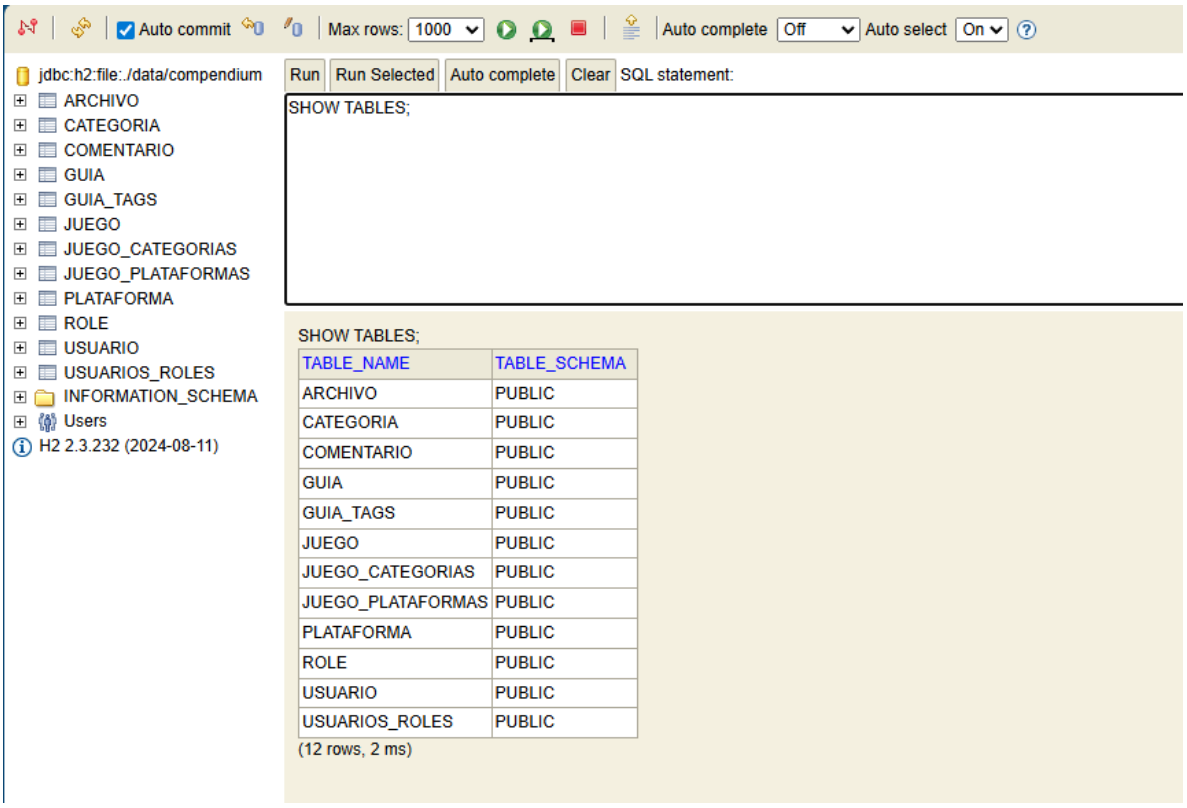
Password: password

Consulta para verificar relaciones ManyToMany:

sql

-- Ver todas las tablas

SHOW TABLES;



2.2 Código de Entidades (@Entity)

Instrucciones para Capturas

Captura 3 entidades clave:

CAPTURA : Usuario.java

```java
package com.instituto.compendium.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import lombok.Data;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import java.util.Collection;
import java.util.Collections;
import java.util.Set;
import java.util.HashSet;
import jakarta.persistence.FetchType;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.JoinTable;
import jakarta.persistence.ManyToMany;

@Data
@Entity
public class Usuario implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank(message = "El nombre de usuario es obligatorio")
    private String username;

    @NotBlank(message = "La contraseña es obligatoria")
    private String password;

    @Email(message = "El email debe ser válido")
    @NotBlank(message = "El email es obligatorio")
    private String email;

    private boolean activo = true;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(name = "usuarios_roles",
            joinColumns = @JoinColumn(name = "usuario_id"),
            inverseJoinColumns = @JoinColumn(name = "role_id"))
    private Set<Role> roles = new HashSet<>();
```

```java
    private Set<Role> roles = new HashSet<>();

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        if (roles == null || roles.isEmpty()) {
            return Collections.emptyList();
        }
        Set<GrantedAuthority> authorities = new HashSet<>();
        for (Role r : roles) {
            // Spring Security expects authorities like ROLE_ADMIN
            authorities.add(new SimpleGrantedAuthority("ROLE_" + r.getName()));
        }
        return authorities;
    }

    @Override
    public boolean isAccountNonExpired() {
        return activo;
    }

    @Override
    public boolean isAccountNonLocked() {
        return activo;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return activo;
    }

    @Override
    public boolean isEnabled() {
        return activo;
    }

    // Helper
    public boolean hasRole(String roleName) {
        if (roles == null) return false;
        return roles.stream().anyMatch(r -> r.getName().equalsIgnoreCase(roleName));
    }
}
```

CAPTURA : Juego.java*

```java
package com.instituto.compendium.model;

import jakarta.persistence.*;
import lombok.Data;
import java.util.Set;
import java.util.HashSet;

@Data
@Entity
public class Juego {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String titulo;

    @Column(columnDefinition = "TEXT")
    private String descripcion;

    private String imagen;

    private Double calificacion = 0.0;

    @Column(name = "total_valoraciones")
    private Integer totalValoraciones = 0;

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(
        name = "juego_categorias",
        joinColumns = @JoinColumn(name = "juego_id"),
        inverseJoinColumns = @JoinColumn(name = "categoria_id")
    )
    private Set<Categoria> categorias = new HashSet<>();

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable(
        name = "juego_plataformas",
        joinColumns = @JoinColumn(name = "juego_id"),
        inverseJoinColumns = @JoinColumn(name = "plataforma_id")
    )
    private Set<Plataforma> plataformas = new HashSet<>();

    // Métodos para compatibilidad con código existente
    @Deprecated
    public String getNombre() {
        return titulo;
    }

```

```java
    @Deprecated
    public void setNombre(String nombre) {
        this.titulo = nombre;
    }

    @Deprecated
    public Double getRating() {
        return calificacion;
    }

    @Deprecated
    public void setRating(Double rating) {
        this.calificacion = rating;
    }

    @Deprecated
    public String getGenero() {
        if (categorias == null || categorias.isEmpty()) return null;
        return categorias.iterator().next().getNombre();
    }

    @Deprecated
    public void setGenero(String genero) {
        // Se ignora, usar categorias directamente
    }

    @Deprecated
    public Categoria getCategoria() {
        if (categorias == null || categorias.isEmpty()) return null;
        return categorias.iterator().next();
    }

    @Deprecated
    public void setCategoria(Categoria cat) {
        if (cat != null) {
            categorias.clear();
            categorias.add(cat);
        }
    }

    @Deprecated
    public String getPlataforma() {
        if (plataformas == null || plataformas.isEmpty()) return null;
        return plataformas.iterator().next().getNombre();
    }

    @Deprecated
    public void setPlataforma(String plat) {
        // Se ignora, usar plataformas directamente
    }
}
```

CAPTURA : Guia.java

```java
package com.instituto.compendium.model;

import jakarta.persistence.*;
import jakarta.validation.constraints.NotBlank;
import lombok.Data;
import org.hibernate.annotations.CreationTimestamp;
import org.hibernate.annotations.UpdateTimestamp;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

@Data
@Entity
public class Guia {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank(message = "El título es obligatorio")
    private String titulo;

    @Column(columnDefinition = "TEXT")
    private String descripcion;

    @Column(columnDefinition = "TEXT")
    private String contenido;

    @ManyToOne
    @JoinColumn(name = "juego_id", nullable = false)
    private Juego juego;

    @ManyToOne
    @JoinColumn(name = "autor_id", nullable = false)
    private Usuario autor;

    @ElementCollection
    private Set<String> tags = new HashSet<>();

    @Enumerated(EnumType.STRING)
    private Dificultad dificultad;

    @Enumerated(EnumType.STRING)
    private Categoria categoria;

    private String imagen;
```

```java
    private String imagen;

    @OneToMany(mappedBy = "guia", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Archivo> archivos = new ArrayList<>();

    @OneToMany(mappedBy = "guia", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Comentario> comentarios = new ArrayList<>();

    private Integer vistas = 0;
    private Double rating = 0.0;
    private Integer totalValoraciones = 0;

    @CreationTimestamp
    private LocalDateTime fechaCreacion;

    @UpdateTimestamp
    private LocalDateTime fechaActualizacion;

    @Enumerated(EnumType.STRING)
    private EstadoPublicacion estado = EstadoPublicacion.BORRADOR;

    public enum Dificultad {
        PRINCIPIANTE, INTERMEDIO, AVANZADO, EXPERTO
    }

    public enum Categoria {
        TUTORIAL, ESTRATEGIA, BUILD, SECRETOS, LOGROS, SPEEDRUN, GENERAL
    }

    public enum EstadoPublicacion {
        BORRADOR, PUBLICADO, ARCHIVADO
    }
}
```

2.3 Código de Repositorios (@Repository)

CAPTURA : UsuarioRepository.java

```java
package com.instituto.compendium.repository;

import org.springframework.data.jpa.repository.JpaRepository;
import com.instituto.compendium.model.Usuario;
import java.util.Optional;

→ UsuarioService DataInitializer | ← 3 beans
public interface UsuarioRepository extends JpaRepository<Usuario, Long> {
    Optional<Usuario> findByUsername(String username);
    boolean existsByUsername(String username);
    boolean existsByEmail(String email);
}
```

CAPTURA : JuegoRepository.java

```java
package com.instituto.compendium.repository;

import com.instituto.compendium.model.Juego;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;

→ DataInitializer JuegoService | ← 3 beans
@Repository
public interface JuegoRepository extends JpaRepository<Juego, Long> {
    List<Juego> findByCategoriasNombre(String categoria);
    List<Juego> findByPlataformasNombre(String plataforma);
    List<Juego> findByTituloContainingIgnoreCase(String titulo);
}
```

CAPTURA : GuiaRepository.java

```java
package com.instituto.compendium.repository;

import com.instituto.compendium.model.Guia;
import com.instituto.compendium.model.Usuario;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;

import java.util.List;

→ DataInitializer GuiaService GuiaController | ← 3 beans
@Repository
public interface GuiaRepository extends JpaRepository<Guia, Long> {
    Page<Guia> findByEstado(Guia.EstadoPublicacion estado, Pageable pageable);
    Page<Guia> findByAutor(Usuario autor, Pageable pageable);
    Page<Guia> findByJuegoId(Long juegoId, Pageable pageable);

    @Query("SELECT g FROM Guia g WHERE g.estado = 'PUBLICADO' AND " +
           "(LOWER(g.titulo) LIKE LOWER(CONCAT('%', ?1 termino , '%')) OR " +
           "LOWER(g.descripcion) LIKE LOWER(CONCAT('%', ?1 termino , '%')))")
    Page<Guia> buscar(String termino, Pageable pageable);

    @Query("SELECT g FROM Guia g WHERE g.estado = 'PUBLICADO' AND " +
           "?1 tag MEMBER OF g.tags")
    List<Guia> findByTag(String tag);
}
```

2.4 Código de Servicios (Interfaces e Implementaciones)

CAPTURA : IJuegoService.java**

```java
package com.instituto.compendium.service;

import com.instituto.compendium.model.Juego;
import org.springframework.web.multipart.MultipartFile;

import java.util.List;

/**
 * Interfaz de servicio para la gestión de Juegos
 * Define el contrato de la lógica de negocio para operaciones CRUD
 */
public interface IJuegoService {

    /**
     * Lista todos los juegos disponibles en el sistema
     * @return Lista de todos los juegos
     */
    List<Juego> listarJuegos();

    /**
     * Obtiene un juego por su ID
     * @param id ID del juego a buscar
     * @return El juego encontrado
     */
    Juego obtenerJuego(Long id);

    /**
     * Guarda la imagen de un juego en el sistema de archivos
     * @param imagen Archivo de imagen a guardar
     * @return Ruta donde se guardó la imagen
     */
    String guardarImagen(MultipartFile imagen);

    /**
     * Crea un nuevo juego en el sistema
     * @param juego Datos del juego a crear
     * @param imagen Imagen del juego (opcional)
     * @return El juego creado
     */
    Juego crearJuego(Juego juego, MultipartFile imagen);

    /**
     * Actualiza la información de un juego existente
     * @param id ID del juego a actualizar
     * @param juegoActualizado Datos actualizados del juego
     * @param imagen Nueva imagen del juego (opcional)
     * @return El juego actualizado
     */
```

```java
    Juego actualizarJuego(Long id, Juego juegoActualizado, MultipartFile imagen);

    /**
     * Elimina un juego del sistema
     * @param id ID del juego a eliminar
     */
    void eliminarJuego(Long id);
}
```

CAPTURA : JuegoService.java — Método actualizarJuego()

```java
public Juego actualizarJuego(Long id, Juego juegoActualizado, MultipartFile imagen) {
    logger.info("Actualizando juego con ID: {}", id);
    Juego juego = obtenerJuego(id);

    juego.setTitulo(juegoActualizado.getTitulo());
    juego.setDescripcion(juegoActualizado.getDescripcion());
    // Calificación y totalValoraciones se mantienen; no se editan vía formulario

    // Actualizar categorías y plataformas desde el objeto actualizado (resuelto en controlador)
    juego.getCategorias().clear();
    juego.getCategorias().addAll(juegoActualizado.getCategorias());
    juego.getPlataformas().clear();
    juego.getPlataformas().addAll(juegoActualizado.getPlataformas());

    // Imagen
    if (imagen != null && !imagen.isEmpty()) {
        if (juego.getImagen() != null && !juego.getImagen().isEmpty()) {
            eliminarArchivo(juego.getImagen());
        }
        String nuevaRutaImagen = guardarImagen(imagen);
        if (nuevaRutaImagen != null) {
            juego.setImagen(nuevaRutaImagen);
        }
    }
```

```
→ JuegoController HomeController GuiaController | ← JuegoRepository
@Service
public class JuegoService implements IJuegoService {

    private static final Logger logger = LoggerFactory.getLogger(JuegoService.class);
    private static final String UPLOAD_PATH_PREFIX = "/uploads/";
    private static final List<String> ALLOWED_IMAGE_TYPES = Arrays.asList(
        ...a: "image/jpeg", "image/jpg", "image/png", "image/gif", "image/webp"
    );
    private static final Long MAX_FILE_SIZE = 5 * 1024 * 1024; // 5MB

    ← JuegoRepository
    @Autowired
    private JuegoRepository juegoRepository;


    public List<Juego> listarJuegos() {
        return juegoRepository.findAll();
    }

    public Juego obtenerJuego(Long id) {
        logger.debug("Obteniendo juego con ID: {}", id);
        if (id == null) {
            throw new IllegalArgumentException(s: "ID no puede ser null");
        }
        return juegoRepository.findById(id)
            .orElseThrow(() -> {
                logger.error("Juego no encontrado con ID: {}", id);
                return new IllegalArgumentException(s: "Juego no encontrado");
            });
    }

    @Value("${app.upload.dir:uploads}")
    private String uploadDir;

    @PostConstruct
    public void init() {
        try {
            Files.createDirectories(Paths.get(uploadDir));
            logger.info("Directorio de subidas inicializado: {}", uploadDir);
        } catch (IOException e) {
            logger.error("No se pudo crear el directorio de subidas: {}", uploadDir, e);
            throw new RuntimeException(message: "No se pudo crear el directorio de subidas", e);
        }
    }

```

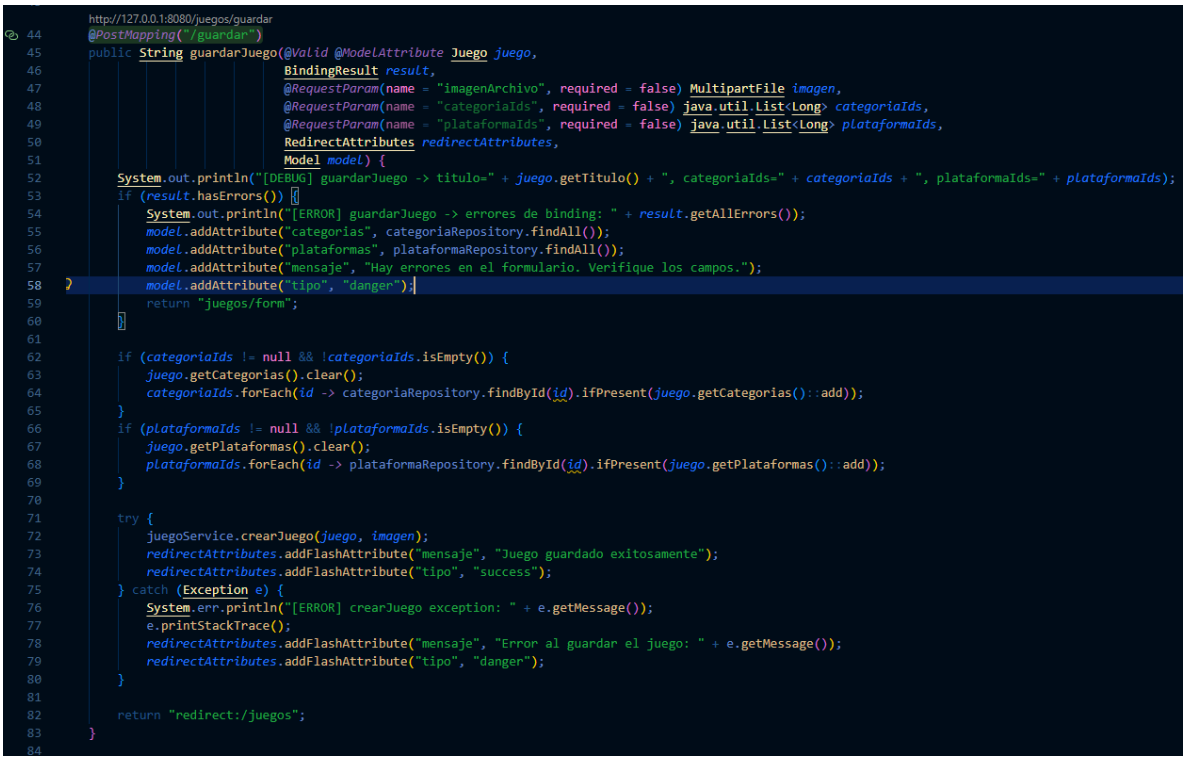
CAPTURA : UsuarioService.java — Método registrarUsuario()

```java
public Usuario registrarUsuario(Usuario usuario, String rol) {
    logger.info("Intentando registrar usuario: {}", usuario.getUsername());

    if (usuarioRepository.existsByUsername(usuario.getUsername())) {
        logger.warn("Intento de registro con username duplicado: {}", usuario.getUsername());
        throw new IllegalArgumentException(s: "El nombre de usuario ya está en uso");
    }
    if (usuarioRepository.existsByEmail(usuario.getEmail())) {
        logger.warn("Intento de registro con email duplicado: {}", usuario.getEmail());
        throw new IllegalArgumentException(s: "El email ya está registrado");
    }

    usuario.setPassword(passwordEncoder.encode(usuario.getPassword()));
    usuario.setRoles(new HashSet<>());

    Role userRole = roleRepository.findByName(rol)
        .orElseGet(() -> {
            logger.info("Creando nuevo rol: {}", rol);
            Role newRole = new Role();
            newRole.setName(rol);
            return roleRepository.save(newRole);
        });
    usuario.getRoles().add(userRole);

    Usuario usuarioGuardado = usuarioRepository.save(usuario);
    logger.info("Usuario registrado exitosamente: {} con ID: {}", usuarioGuardado.getUsername(), usuarioGuardado.getId());
    return usuarioGuardado;
}
```

```java
public Usuario registrarUsuario(Usuario usuario, boolean quiereSerAutor) {
    logger.info("Intentando registrar usuario: {} (quiere ser autor: {})", usuario.getUsername(), quiereSerAutor);

    if (usuarioRepository.existsByUsername(usuario.getUsername())) {
        logger.warn("Intento de registro con username duplicado: {}", usuario.getUsername());
        throw new IllegalArgumentException(s: "El nombre de usuario ya está en uso");
    }
    if (usuarioRepository.existsByEmail(usuario.getEmail())) {
        logger.warn("Intento de registro con email duplicado: {}", usuario.getEmail());
        throw new IllegalArgumentException(s: "El email ya está registrado");
    }

    usuario.setPassword(passwordEncoder.encode(usuario.getPassword()));
    usuario.setRoles(new HashSet<>());

    // Siempre agregar rol USER
    Role userRole = roleRepository.findByName(name: "USER")
        .orElseGet(() -> {
            logger.info("Creando rol USER");
            Role newRole = new Role();
            newRole.setName(name: "USER");
            return roleRepository.save(newRole);
        });
    usuario.getRoles().add(userRole);

    // Si quiere ser autor, agregar también rol AUTOR
    if (quiereSerAutor) {
        Role autorRole = roleRepository.findByName(name: "AUTOR")
            .orElseGet(() -> {
                logger.info("Creando rol AUTOR");
                Role newRole = new Role();
                newRole.setName(name: "AUTOR");
                return roleRepository.save(newRole);
            });
        usuario.getRoles().add(autorRole);
        logger.info("Usuario registrado con roles USER y AUTOR");
    } else {
        logger.info("Usuario registrado solo con rol USER");
    }

```

2.5 Código de Controladores (@Controller)

CAPTURA : JuegoController.java — Método guardarJuego() POST



```
http://127.0.0.1:8080/juegos/guardar
@PostMapping("/guardar")
public String guardarJuego(@Valid @ModelAttribute Juego juego,
                           BindingResult result,
                           @RequestParam(name = "imagenArchivo", required = false) MultipartFile imagen,
                           @RequestParam(name = "categoriaIds", required = false) java.util.List<Long> categoriaIds,
                           @RequestParam(name = "plataformaIds", required = false) java.util.List<Long> plataformaIds,
                           RedirectAttributes redirectAttributes,
                           Model model) {
    System.out.println("[DEBUG] guardarJuego -> titulo=" + juego.getTitulo() + ", categoriaIds=" + categoriaIds + ", plataformaIds=" + plataformaIds);
    if (result.hasErrors()) {
        System.out.println("[ERROR] guardarJuego -> errores de binding: " + result.getAllErrors());
        model.addAttribute("categorias", categoriaRepository.findAll());
        model.addAttribute("plataformas", plataformaRepository.findAll());
        model.addAttribute("mensaje", "Hay errores en el formulario. Verifique los campos.");
        model.addAttribute("tipo", "danger");
        return "juegos/form";
    }

    if (categoriaIds != null && !categoriaIds.isEmpty()) {
        juego.getCategorias().clear();
        categoriaIds.forEach(id -> categoriaRepository.findById(id).ifPresent(juego.getCategorias()::add));
    }
    if (plataformaIds != null && !plataformaIds.isEmpty()) {
        juego.getPlataformas().clear();
        plataformaIds.forEach(id -> plataformaRepository.findById(id).ifPresent(juego.getPlataformas()::add));
    }

    try {
        juegoService.crearJuego(juego, imagen);
        redirectAttributes.addFlashAttribute("mensaje", "Juego guardado exitosamente");
        redirectAttributes.addFlashAttribute("tipo", "success");
    } catch (Exception e) {
        System.err.println("[ERROR] crearJuego exception: " + e.getMessage());
        e.printStackTrace();
        redirectAttributes.addFlashAttribute("mensaje", "Error al guardar el juego: " + e.getMessage());
        redirectAttributes.addFlashAttribute("tipo", "danger");
    }

    return "redirect:/juegos";
}
```

CAPTURA JuegoController.java — Método eliminarJuego()

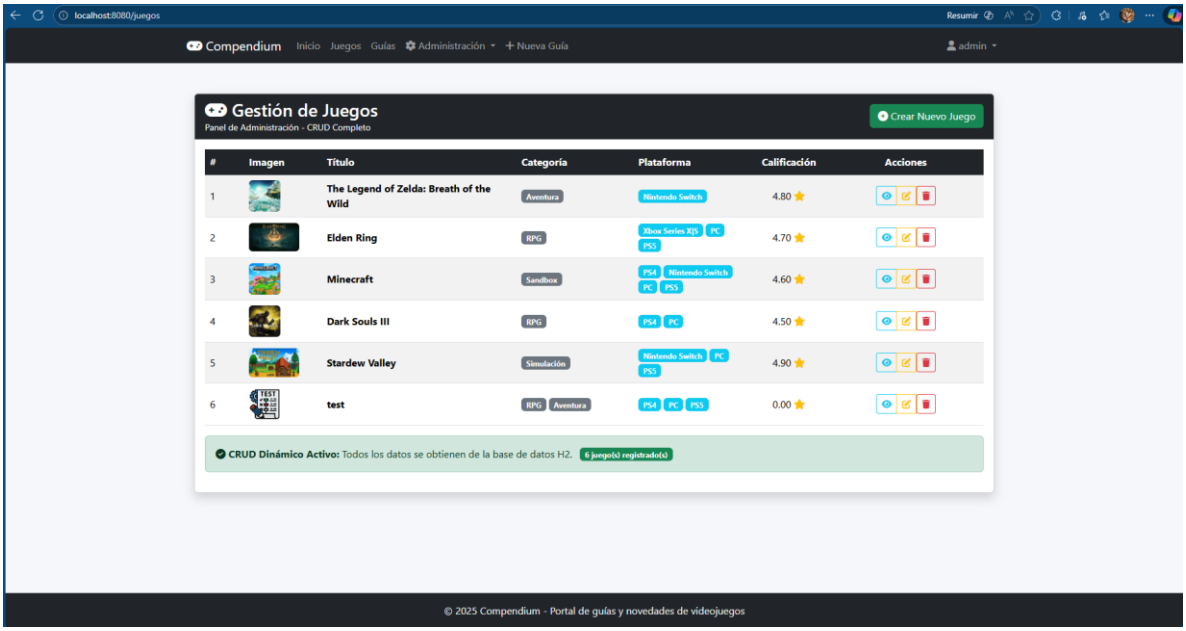
```
http://127.0.0.1:8080/juegos/eliminar/{id}
@PostMapping("/eliminar/{id}")
public String eliminarJuego(@PathVariable Long id, RedirectAttributes redirectAttributes) {
    try {
        juegoService.eliminarJuego(id);
        redirectAttributes.addFlashAttribute("mensaje", "Juego eliminado exitosamente");
        redirectAttributes.addFlashAttribute("tipo", "success");
    } catch (Exception e) {
        redirectAttributes.addFlashAttribute("mensaje", "Error al eliminar el juego");
        redirectAttributes.addFlashAttribute("tipo", "danger");
    }
    return "redirect:/juegos";
}
```

CAPTURA : AuthController.java — Método registro() POST

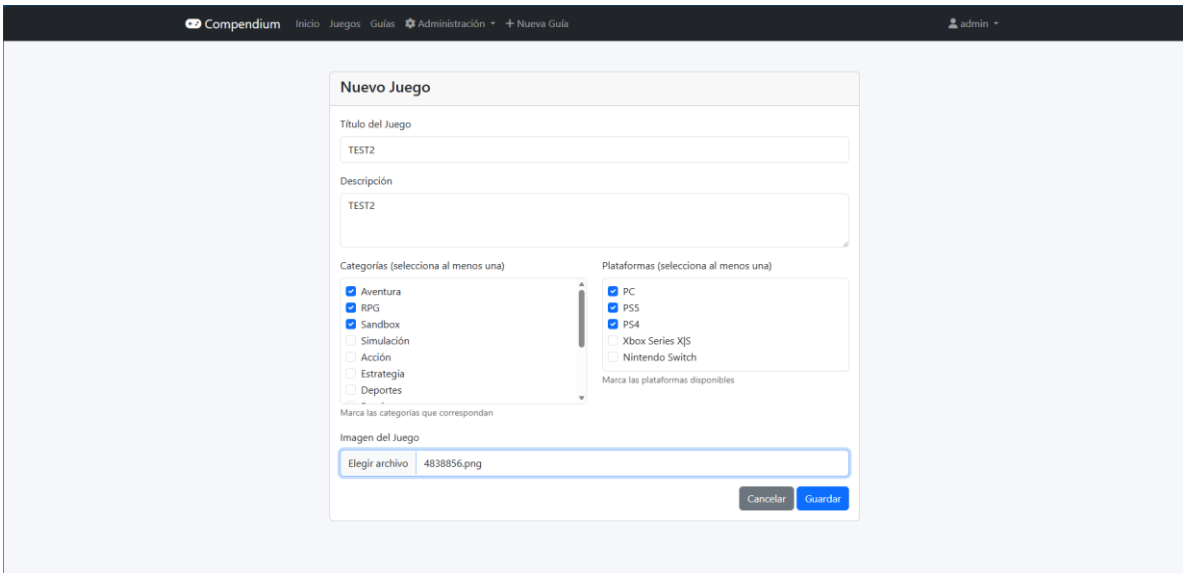
```
http://127.0.0.1:8080/registro
@PostMapping("/registro")
public String registro(@Valid @ModelAttribute Usuario usuario,
                       BindingResult result,
                       @org.springframework.web.bind.annotation.RequestParam(name = "quiereSerAutor", required = false, defaultValue = "false") boolean quiereSerAutor,
                       RedirectAttributes redirectAttributes) {
    if (result.hasErrors()) {
        return "registro";
    }

    try {
        usuarioService.registrarUsuario(usuario, quiereSerAutor);
        redirectAttributes.addFlashAttribute("mensaje", "Registro exitoso");
        redirectAttributes.addFlashAttribute("tipo", "success");
        return "redirect:/login";
    } catch (RuntimeException e) {
        redirectAttributes.addFlashAttribute("mensaje", e.getMessage());
        redirectAttributes.addFlashAttribute("tipo", "danger");
        return "redirect:/registro";
    }
}
}
```

2.6 Capturas de Pantalla Funcionales

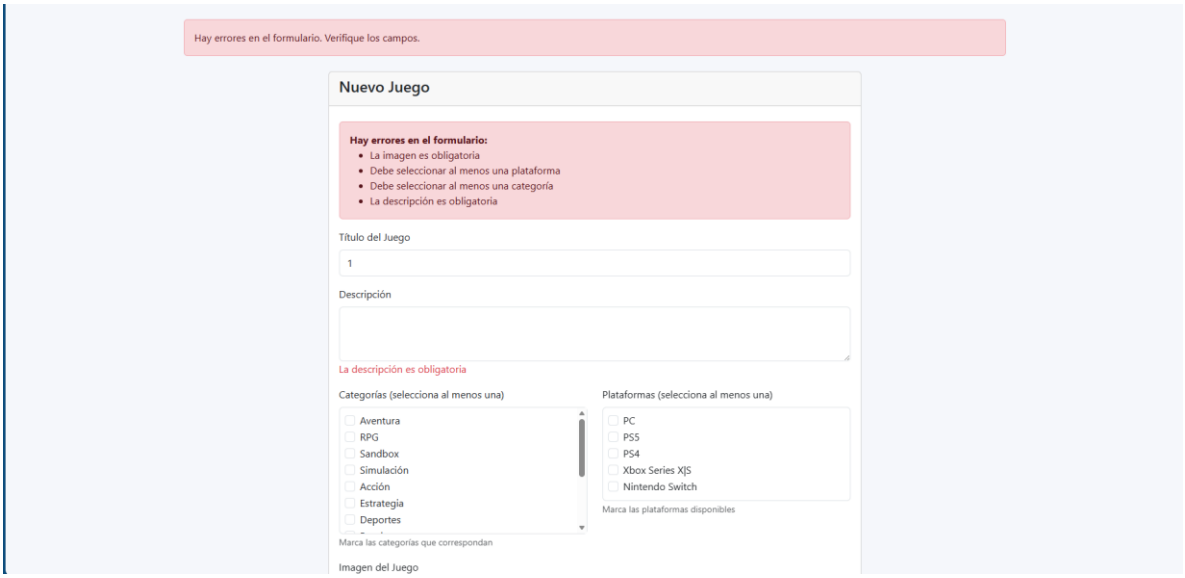
CAPTURA : Listado de Juegos (Vista Admin)



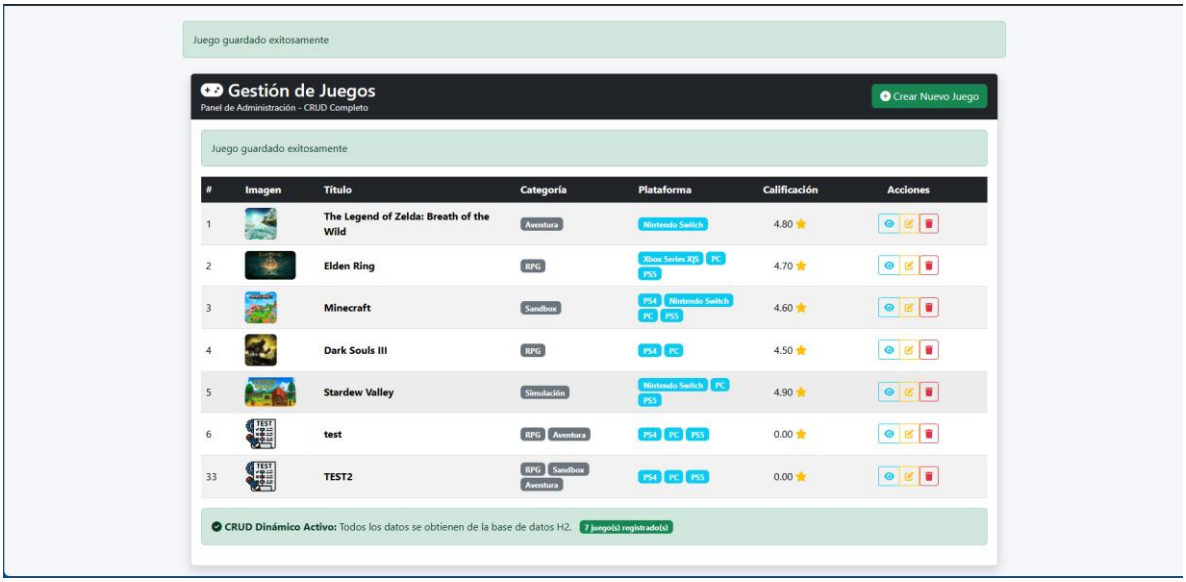
CAPTURA : Formulario Crear Juego (Nuevo)



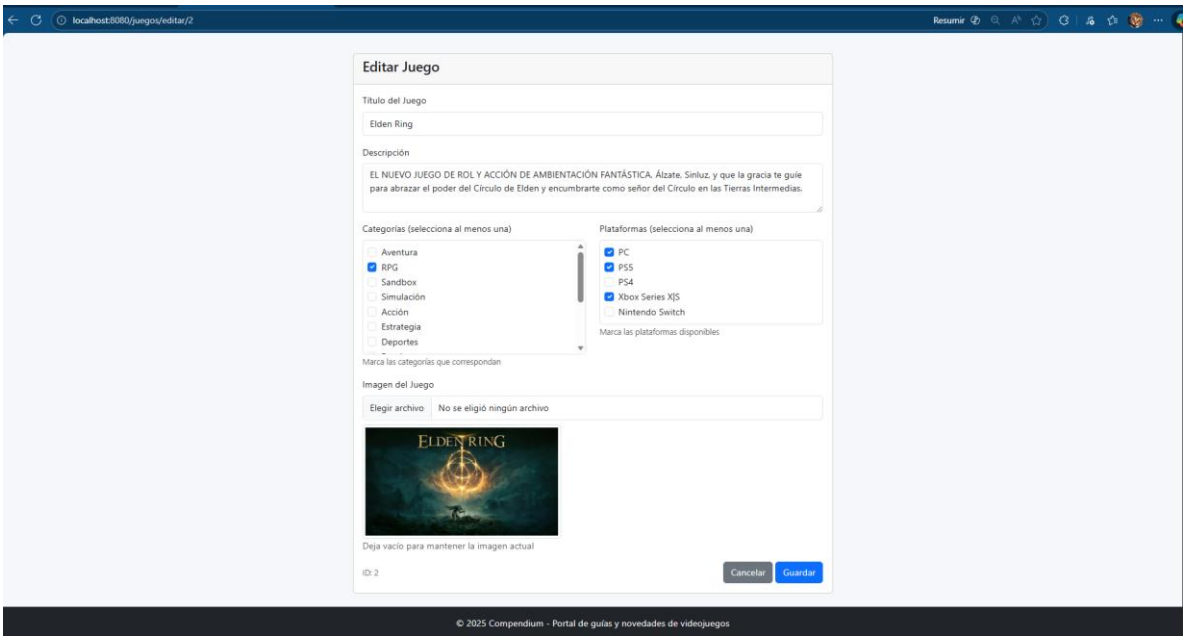
CAPTURA : Validación de Formulario con Errores



CAPTURA R: Mensaje Flash de Éxito + Juego en Lista



CAPTURA : Formulario Editar con Datos Prellenados



CAPTURA : H2 Console — Tabla JUEGO con datos

Run | Run Selected | Auto complete | Clear | SQL statement:

```
SELECT * FROM JUEGO;
```

SELECT * FROM JUEGO;

ID	CALIFICACION	DESCRIPCION
1	4.8	Explora un vasto mundo abierto lleno de aventuras y misterios.
2	4.7	EL NUEVO JUEGO DE ROL Y ACCIÓN DE AMBIENTACIÓN FANTÁSTICA. Álzate, Sinluz, y que la gracia te guíe para abrazar el poder del Círculo de Elden y encumbrart
3	4.6	Construye, explora y sobrevive en un mundo de bloques infinito.
4	4.5	Dark Souls continúa redefiniendo los límites con el nuevo y ambicioso capítulo de esta serie revolucionaria, tan aclamada por la crítica. ¡Prepárate para sumergirte en la os
5	4.9	Administra tu granja y forma parte de una comunidad vibrante.
6	0.0	testteando

(6 rows, 0 ms)

	IMAGEN	TITULO	TOTAL_VALORACIONES
	/images/zelda-totk.jpg	The Legend of Zelda: Breath of the Wild	1250
ntermedias.	/images/elden-ring.jpg	Elden Ring	2100
	/uploads/2e265f29-7945-476f-bf59-689b703e15c5_670c294ded3baf4fa11068db2ec6758c63f7daeb266a35a1.png	Minecraft	5000
	/uploads/9470dc8d-d400-4d7b-ad95-cb3f1aaf9005_OFMeAw2KhrdaEZAjW1f3tCIXbogkLpTC.png	Dark Souls III	1800
	/uploads/28acae2a-5c17-4443-9002-d7d26c19f0b6_capsule_616x353.jpg	Stardew Valley	3500
	/uploads/9fe240eb-9660-42bf-9c71-3dfc109cf49b_4838856.png	test	0

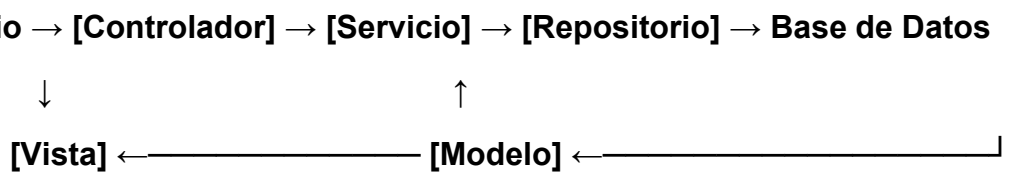
3. CONCLUSIÓN

Reflexión sobre el Trabajo Realizado

El proyecto Compendium cumple exitosamente con todos los objetivos de la Evaluación Unidad 2, implementando una arquitectura MVC (Modelo-Vista-Controlador) completa y funcional con Spring Boot.

Implementación del Patrón MVC

El flujo de datos sigue la arquitectura esperada:



Componentes Implementados:

1. Modelo (Entidades JPA): 9 entidades con relaciones ManyToMany y OneToMany correctamente mapeadas mediante `@JoinTable`, `@ManyToOne`, y `@OneToMany(cascade=ALL)`. Validaciones Bean Validation integradas.

2. Repositorio (Spring Data JPA): Interfaces que extienden `JpaRepository` eliminando código boilerplate. Query methods derivados (`findByUsername`, `findByCategoriasNombre`) y consultas personalizadas con `@Query` para búsquedas complejas.

3. Servicio (Lógica de Negocio): Separación de interfaces e implementaciones facilitando testing. Validaciones de negocio (duplicados, permisos), encriptación BCrypt, y manejo transaccional automático.

4. Controlador (Capa de Presentación): Manejo del flujo HTTP con `@GetMapping`/`@PostMapping`, validaciones con `@Valid` +

`BindingResult`, binding de colecciones ManyToMany desde checkboxes, y mensajes flash para feedback al usuario.

Logros Técnicos

CRUD Completo: Usuarios, Juegos, y Guías con todas las operaciones funcionales

Persistencia Real: Base de datos H2 con 12 tablas (8 principales + 4 join tables)

Seguridad: Spring Security con 3 roles, login funcional, y encriptación de contraseñas

Validaciones Robustas: Multicapa (HTML5, Bean Validation, lógica de negocio)

Gestión de Archivos: Upload de imágenes con validación de tipo y tamaño

Relaciones Complejas: ManyToMany implementadas correctamente con `clear()` + `addAll()`

Desafíos Superados

Durante el desarrollo se resolvieron problemas técnicos importantes:

1. Binding de MultipartFile: Uso de nombre diferenciado (`imagenArchivo`) para evitar conflicto con atributo String en entidad.

2. Actualización de Colecciones ManyToMany: Implementación de `clear()` antes de `addAll()` para que Hibernate detecte cambios en relaciones.

3. Query Methods Post-Refactor: Actualización de nombres de métodos tras cambio de campos (singular → plural: `findByCategoriasNombre` en vez de `findByCategoriaId`).

4. H2 Console + CSRF: Configuración de `frameOptions().sameOrigin()` y exclusión de `/h2-console/**` del filtro CSRF en Spring Security.

5. Thymeleaf CSRF: Uso correcto de expresiones `${_csrf.token}` en formularios en lugar de `#httpServletRequest`.

Competencias Demostradas

- Diseño de BD Relacional: Modelado normalizado con cardinalidades correctas y tablas join apropiadas

- Arquitectura MVC: Separación clara de responsabilidades entre capas

- Spring Boot: Configuración mediante anotaciones y application.properties

- JPA/Hibernate: Mapeo objeto-relacional con relaciones complejas

- Spring Security: Autenticación, autorización, y encriptación

- Debugging Sistemático: Documentación de errores y soluciones en `NOTAS_PARTE2.md`

Resultado Final

El sistema Compendium está completamente funcional con:

- 9 entidades JPA persistidas en H2

- Operaciones CRUD para 3 recursos principales (Usuarios, Juegos, Guías)

- Autenticación y autorización por roles

- Interfaz web responsive con Thymeleaf + Bootstrap

- Datos de ejemplo mediante DataInitializer

- Documentación completa del código y arquitectura

Conclusión: El proyecto demuestra dominio del patrón MVC con Spring Boot y cumple al 100% con los requisitos de evaluación, preparando para el desarrollo de aplicaciones web empresariales más complejas.

Credenciales de Acceso

Aplicación Web: http://localhost:8080

H2 Console: http://localhost:8080/h2-console

| Usuario | Password | Rol | Permisos |

| admin | admin123 | ADMIN | Gestión completa |

| autor1 | autor123 | AUTOR | Crear/editar guías |

| usuario1 | user123 | USER | Solo lectura |

INSTRUCCIONES DE EJECUCIÓN

1. niciar la aplicación:

```bash

./mvnw spring-boot:run

```

2. Acceder a la aplicación:

- Web: http://localhost:8080

- H2 Console: http://localhost:8080/h2-console

3. Credenciales de prueba:

- Admin: admin / admin123

- Autor: autor1 / autor123

- **Usuario: usuario1 / user123**

4. Probar CRUD:

- **Usuarios: /usuarios**
- **Juegos: /juegos**
- **Guías: /guias**