MasterClass: Tecnologías para un Sistema CRUD de Login

Contenido

> **Objetivo de aprendizaje**
>
> Al finalizar esta masterclass serás capaz de implementar un sistema completo de gestión de usuarios con autenticación segura usando Spring Boot, JPA, Hibernate, BCrypt y Lombok.

Visión General del Stack Tecnológico

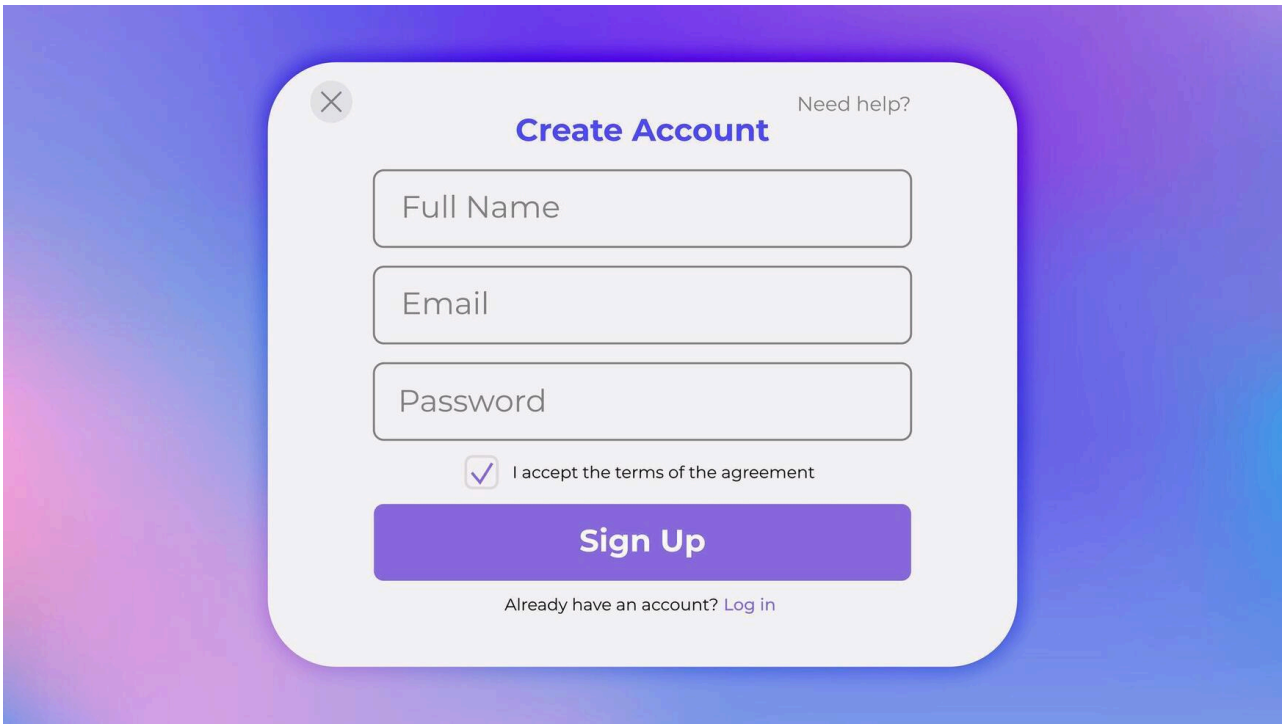


Figura 1 Stack tecnológico para sistema CRUD de usuarios

Backend Framework

- Spring Boot 3.x
- Spring Data JPA
- Spring Security Crypto

Persistencia

- Hibernate ORM
- MySQL 8.x
- JPA 3.0

---

1. Spring Boot - El Framework Principal

¿Qué es Spring Boot?

Spring Boot es un framework que simplifica la creación de aplicaciones Spring eliminando la configuración manual. Proporciona:

- **Auto-configuración**: Detecta las dependencias y configura automáticamente
- **Servidor embebido**: Tomcat incluido, no necesitas desplegarlo externamente
- **Starters**: Dependencias agrupadas por funcionalidad

En la práctica profesional

Spring Boot es el estándar de la industria para aplicaciones Java empresariales. Más del 60% de las ofertas de trabajo Java piden experiencia con Spring Boot.

Configuración del Proyecto

1.1 Dependencias Maven (pom.xml)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
         https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.2.0</version>
        <relativePath/>
    </parent>

    <groupId>com.ejemplo</groupId>
    <artifactId>gestion-usuarios</artifactId>
    <version>1.0.0</version>
    <name>Sistema de Gestión de Usuarios</name>

    <properties>
        <java.version>17</java.version>
    </properties>

    <dependencies>
        <!-- Spring Data JPA - Acceso a datos con JPA/Hibernate -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>

        <!-- MySQL Connector - Driver de base de datos -->
        <dependency>
            <groupId>com.mysql</groupId>
            <artifactId>mysql-connector-j</artifactId>
            <scope>runtime</scope>
        </dependency>

        <!-- Spring Security Crypto - Para BCrypt -->
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-crypto</artifactId>
        </dependency>

        <!-- Lombok - Reducción de código boilerplate -->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
            <optional>true</optional>
        </dependency>

        <!-- Testing -->
        <dependency>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
                <configuration>
                    <excludes>
                        <exclude>
                            <groupId>org.projectlombok</groupId>
                            <artifactId>lombok</artifactId>
                        </exclude>
                    </excludes>
                </configuration>
            </plugin>
        </plugins>
    </build>
</project>
```

1.2 Clase Principal de Aplicación

```
package com.ejemplo.gestionusuarios;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

/**
 * Clase principal que inicia la aplicación Spring Boot.
 *
 * @SpringBootApplication combina tres anotaciones:
 * - @Configuration: Define la clase como fuente de configuración
 * - @EnableAutoConfiguration: Activa la auto-configuración de Spring Boot
 * - @ComponentScan: Escanea componentes en este paquete y subpaquetes
 */
@SpringBootApplication
public class GestionUsuariosApplication {

    public static void main(String[] args) {
        SpringApplication.run(GestionUsuariosApplication.class, args);
    }
}
```

2. Hibernate - Configuración del ORM

¿Qué es Hibernate?

Hibernate es la implementación más popular de JPA (Java Persistence API). Actúa como puente entre los objetos Java y las tablas de la base de datos.

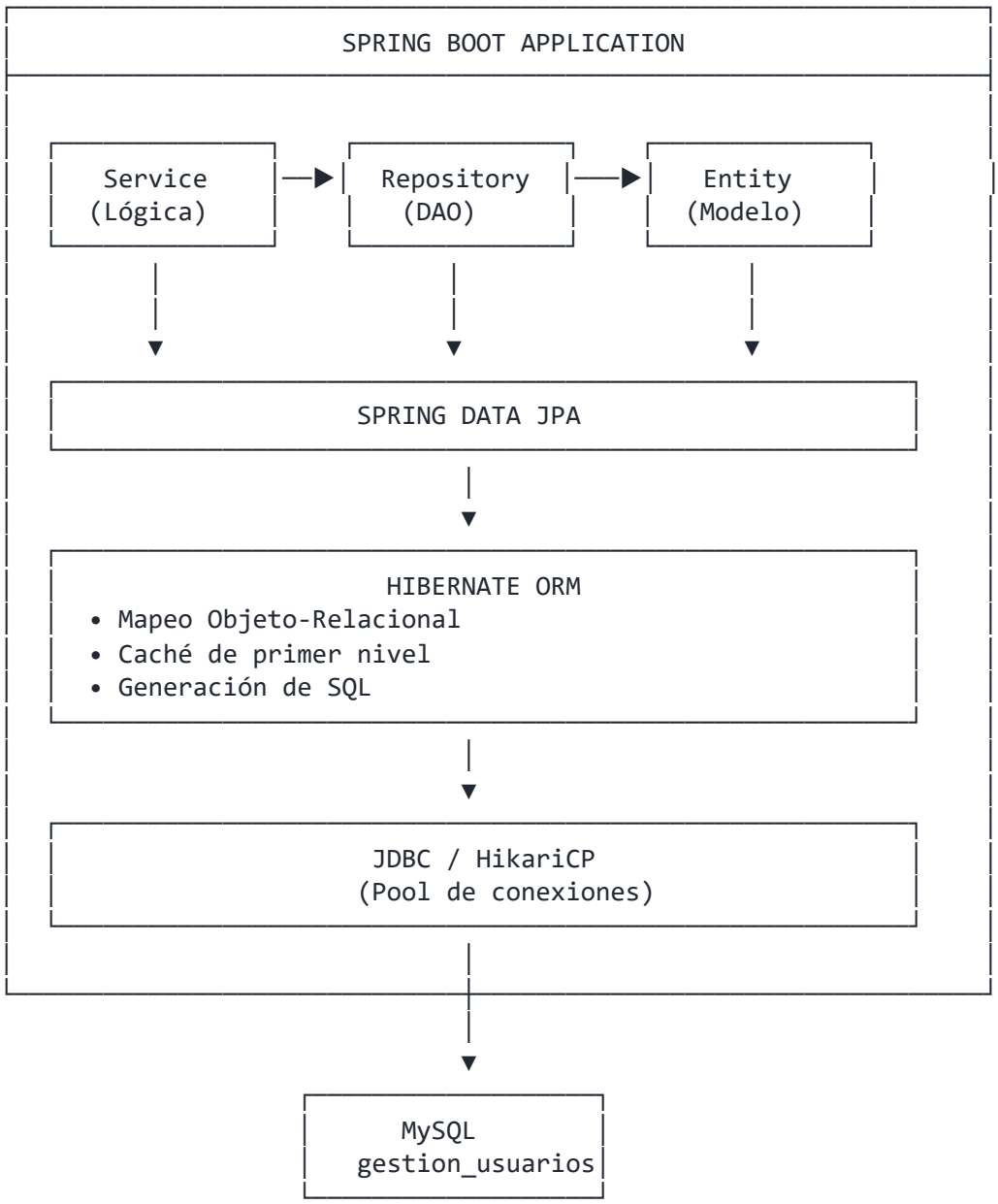
> **Nota técnica**
>
> JPA es la especificación (interfaz), Hibernate es la implementación. Spring Data JPA usa Hibernate internamente por defecto.

2.1 Configuración (application.properties)

```
# ================================================
# CONFIGURACIÓN DE BASE DE DATOS
# ================================================

# URL de conexión a MySQL
# Formato: jdbc:mysql://host:puerto/nombre_base_datos
spring.datasource.url=jdbc:mysql://localhost:3306/gestion_usuarios?useSSL=false&ser

# Credenciales de acceso
spring.datasource.username=root
spring.datasource.password=tu_password

# Driver JDBC de MySQL
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# ================================================
# CONFIGURACIÓN DE HIBERNATE/JPA
# ================================================

# Dialecto SQL específico para MySQL 8
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect

# Estrategia de generación del esquema:
# - none: No hace nada (producción)
# - validate: Valida el esquema, no lo modifica
# - update: Actualiza el esquema si hay cambios (desarrollo)
# - create: Crea el esquema, destruyendo datos previos
# - create-drop: Crea al iniciar, borra al cerrar
spring.jpa.hibernate.ddl-auto=validate

# Mostrar las consultas SQL en consola (solo desarrollo)
spring.jpa.show-sql=true

# Formatear las consultas SQL para mejor legibilidad
spring.jpa.properties.hibernate.format_sql=true

# Mostrar los parámetros de las consultas (solo desarrollo)
logging.level.org.hibernate.orm.jdbc.bind=TRACE

# ================================================
# CONFIGURACIÓN DEL POOL DE CONEXIONES (HikariCP)
# ================================================

# Número máximo de conexiones en el pool
spring.datasource.hikari.maximum-pool-size=10

# Número mínimo de conexiones inactivas
spring.datasource.hikari.minimum-idle=5

# Tiempo máximo de espera para obtener conexión (ms)
spring.datasource.hikari.connection-timeout=30000

# Tiempo máximo que una conexión puede estar inactiva (ms)
spring.datasource.hikari.idle-timeout=600000
```

> ⚠ **Importante**
>
> **NUNCA uses `ddl-auto=create` o `create-drop` en producción**. Esto borraría todos los datos de la base de datos cada vez que reinicies la aplicación.

2.2 Diagrama de Arquitectura



3. Lombok - Reducción de Código Repetitivo

¿Qué es Lombok?

Lombok es una librería que genera automáticamente código repetitivo (boilerplate) en tiempo de compilación mediante anotaciones.

> **Sabías que...**
>
> Sin Lombok, una entidad JPA con 7 campos requeriría escribir aproximadamente 150 líneas de código para getters, setters, constructores, equals, hashCode y toString. Con Lombok: solo 20 líneas.

3.1 Anotaciones Principales de Lombok

Anotación	Genera
`@Getter`	Métodos getter para todos los campos
`@Setter`	Métodos setter para todos los campos
`@NoArgsConstructor`	Constructor sin argumentos
`@AllArgsConstructor`	Constructor con todos los argumentos
`@RequiredArgsConstructor`	Constructor con campos `final` o `@NonNull`
`@ToString`	Método `toString()`
`@EqualsAndHashCode`	Métodos `equals()` y `hashCode()`
`@Data`	Combina: @Getter, @Setter, @ToString, @EqualsAndHashCode, @RequiredArgsConstructor
`@Builder`	Patrón Builder para crear objetos
`@Slf4j`	Logger SLF4J automático

3.2 Comparativa: Con y Sin Lombok

Sin Lombok (150+ líneas) | **Con Lombok (20 líneas)**

```
public class Usuario {
    private Long id;
    private String username;
    private String email;
    private String passwordHash;
    private Boolean activo;
    private LocalDateTime fechaCreacion;
    private LocalDateTime fechaActualizacion;

    // Constructor vacío
    public Usuario() {}

    // Constructor completo
    public Usuario(Long id, String username, String email,
                   String passwordHash, Boolean activo,
                   LocalDateTime fechaCreacion,
                   LocalDateTime fechaActualizacion) {
        this.id = id;
        this.username = username;
        this.email = email;
        this.passwordHash = passwordHash;
        this.activo = activo;
        this.fechaCreacion = fechaCreacion;
        this.fechaActualizacion = fechaActualizacion;
    }

    // Getters
    public Long getId() { return id; }
    public String getUsername() { return username; }
    public String getEmail() { return email; }
    public String getPasswordHash() { return passwordHash; }
    public Boolean getActivo() { return activo; }
    public LocalDateTime getFechaCreacion() { return fechaCreacion; }
    public LocalDateTime getFechaActualizacion() { return fechaActualizacion; }

    // Setters
    public void setId(Long id) { this.id = id; }
    public void setUsername(String username) { this.username = username; }
    public void setEmail(String email) { this.email = email; }
    public void setPasswordHash(String passwordHash) { this.passwordHash = passw
    public void setActivo(Boolean activo) { this.activo = activo; }
    public void setFechaCreacion(LocalDateTime fechaCreacion) { this.fechaCreaci
    public void setFechaActualizacion(LocalDateTime fechaActualizacion) { this.f

    // equals, hashCode, toString... (50+ líneas más)
}
```

4. Spring Data JPA - Repositorios Inteligentes

¿Qué es Spring Data JPA?

Spring Data JPA simplifica el acceso a datos eliminando la necesidad de escribir implementaciones de repositorios. Solo defines una interfaz y Spring genera la implementación automáticamente.

4.1 Creación de Repositorios

```java
package com.ejemplo.gestionusuarios.repository;

import com.ejemplo.gestionusuarios.entity.Usuario;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.List;
import java.util.Optional;

/**
 * Repositorio para operaciones CRUD de Usuario.
 *
 * JpaRepository<Usuario, Long> proporciona:
 * - save(entity): Guardar/actualizar
 * - findById(id): Buscar por ID
 * - findAll(): Obtener todos
 * - deleteById(id): Eliminar por ID
 * - count(): Contar registros
 * - existsById(id): Verificar existencia
 */
@Repository
public interface UsuarioRepository extends JpaRepository<Usuario, Long> {

    // ========================================
    // QUERY METHODS - Derivados del nombre
    // ========================================

    /**
     * Busca usuario por username exacto.
     * Spring genera: SELECT * FROM usuarios WHERE username = ?
     */
    Optional<Usuario> findByUsername(String username);

    /**
     * Busca usuario por email exacto.
     * Spring genera: SELECT * FROM usuarios WHERE email = ?
     */
    Optional<Usuario> findByEmail(String email);

    /**
     * Lista todos los usuarios activos.
     * Spring genera: SELECT * FROM usuarios WHERE activo = true
     */
    List<Usuario> findByActivoTrue();

    /**
     * Lista todos los usuarios inactivos.
     * Spring genera: SELECT * FROM usuarios WHERE activo = false
     */
    List<Usuario> findByActivoFalse();

    /**
     * Verifica si existe un username.
     * Spring genera: SELECT COUNT(*) > 0 FROM usuarios WHERE username = ?
```

```
     */
    boolean existsByUsername(String username);

    /**
     * Verifica si existe un email.
     * Spring genera: SELECT COUNT(*) > 0 FROM usuarios WHERE email = ?
     */
    boolean existsByEmail(String email);

    /**
     * Busca usuarios cuyo username contenga el texto (LIKE).
     * Spring genera: SELECT * FROM usuarios WHERE username LIKE %?%
     */
    List<Usuario> findByUsernameContainingIgnoreCase(String texto);

    // ========================================
    // QUERIES PERSONALIZADAS CON @Query
    // ========================================

    /**
     * Busca usuarios activos ordenados por fecha de creación.
     * Usa JPQL (Java Persistence Query Language).
     */
    @Query("SELECT u FROM Usuario u WHERE u.activo = true ORDER BY u.fechaCreacion
    List<Usuario> findUsuariosActivosOrdenados();

    /**
     * Busca usuarios por dominio de email.
     * Usa JPQL con parámetro nombrado.
     */
    @Query("SELECT u FROM Usuario u WHERE u.email LIKE %:dominio")
    List<Usuario> findByDominioEmail(@Param("dominio") String dominio);

    /**
     * Cuenta usuarios activos.
     */
    @Query("SELECT COUNT(u) FROM Usuario u WHERE u.activo = true")
    long countUsuariosActivos();

    /**
     * Query nativa SQL (cuando JPQL no es suficiente).
     */
    @Query(value = "SELECT * FROM usuarios WHERE DATEDIFF(NOW(), fecha_creacion) <=
           nativeQuery = true)
    List<Usuario> findUsuariosRecientes(@Param("dias") int dias);
}
```

4.2 Convenciones de Query Methods

Tabla 1 Palabras clave para Query Methods

Palabra Clave	Ejemplo	SQL Generad
`findBy`	`findByUsername(String u)`	`WHERE userna = ?`
`findByXxxAndYyy`	`findByUsernameAndActivo(String u, Boolean a)`	`WHERE userna = ? AND activ = ?`
`findByXxxOrYyy`	`findByUsernameOrEmail(String u, String e)`	`WHERE userna = ? OR email ?`
`findByXxxLike`	`findByUsernameLike(String u)`	`WHERE userna LIKE ?`
`findByXxxContaining`	`findByUsernameContaining(String u)`	`WHERE userna LIKE %?%`
`findByXxxStartingWith`	`findByUsernameStartingWith(String u)`	`WHERE userna LIKE ?%`
`findByXxxEndingWith`	`findByUsernameEndingWith(String u)`	`WHERE userna LIKE %?`
`findByXxxIgnoreCase`	`findByUsernameIgnoreCase(String u)`	`WHERE LOWER(usernam = LOWER(?)`
`findByXxxOrderByYyy`	`findByActivoOrderByFechaCreacionDesc(Boolean a)`	`WHERE activo ? ORDER BY fecha_creacic DESC`
`countBy`	`countByActivo(Boolean a)`	`SELECT COUNT(*) WHEF activo = ?`

Palabra Clave	Ejemplo	SQL Generad
`existsBy`	`existsByUsername(String u)`	`SELECT COUNT(*) > 0 WHERE usernam = ?`
`deleteBy`	`deleteByActivo(Boolean a)`	`DELETE WHERE activo = ?`

5. BCrypt - Seguridad de Contraseñas

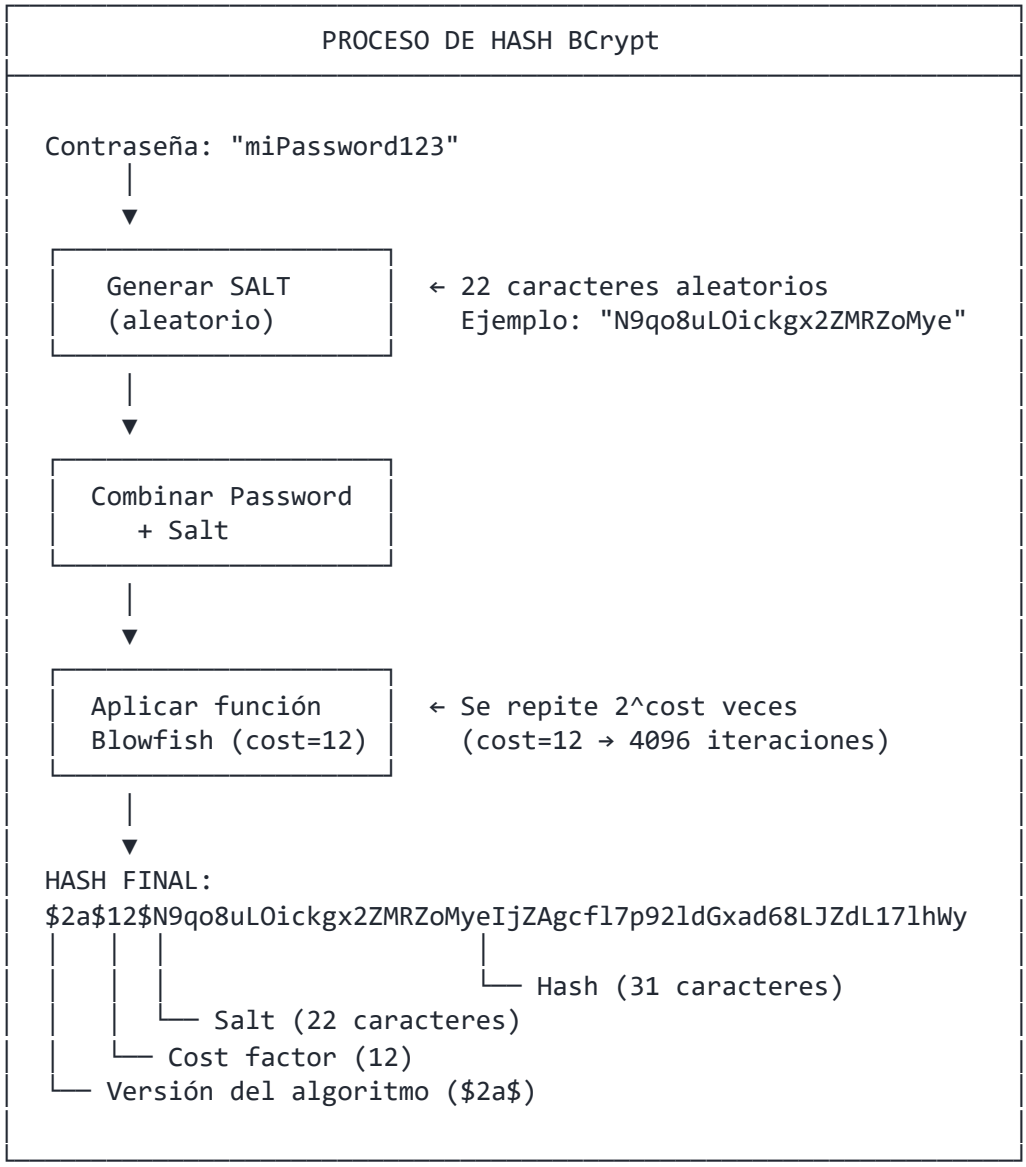
¿Por qué BCrypt?

> ⚠ **Atención**
>
> **NUNCA almacenes contraseñas en texto plano**. Si un atacante accede a tu base de datos, tendría acceso directo a todas las contraseñas.

BCrypt es un algoritmo de hashing diseñado específicamente para contraseñas:

- **Salting automático**: Añade datos aleatorios únicos a cada contraseña
- **Resistente a ataques de fuerza bruta**: Es intencionalmente lento
- **Factor de coste configurable**: Puedes aumentar la dificultad con el tiempo

5.1 Cómo Funciona BCrypt



5.2 Implementación con Spring Security Crypto

```java
package com.ejemplo.gestionusuarios.util;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Component;

/**
 * Utilidad para manejo seguro de contraseñas con BCrypt.
 */
@Component
public class PasswordUtil {

    // Strength = 12 (factor de coste). Valores recomendados: 10-14
    // Mayor valor = más seguro pero más lento
    private static final BCryptPasswordEncoder encoder = new BCryptPasswordEncoder(

    /**
     * Hashea una contraseña en texto plano.
     *
     * @param rawPassword Contraseña en texto plano
     * @return Hash BCrypt de la contraseña
     *
     * Ejemplo:
     *   Input:  "password123"
     *   Output: "$2a$12$N9qo8uLOickgx2ZMRZoMyeIjZAgcfl7p92ldGxad68LJZdL17lhWy"
     */
    public String hashPassword(String rawPassword) {
        return encoder.encode(rawPassword);
    }

    /**
     * Verifica si una contraseña coincide con su hash.
     *
     * @param rawPassword Contraseña en texto plano introducida por el usuario
     * @param hashedPassword Hash almacenado en la base de datos
     * @return true si coinciden, false si no
     *
     * Ejemplo:
     *   verifyPassword("password123", "$2a$12$N9qo8u...") → true
     *   verifyPassword("wrongpass", "$2a$12$N9qo8u...") → false
     */
    public boolean verifyPassword(String rawPassword, String hashedPassword) {
        return encoder.matches(rawPassword, hashedPassword);
    }

    /**
     * Genera un hash BCrypt usando el encoder estático.
     * Útil para generar hashes desde la consola o scripts.
     */
    public static String generateHash(String password) {
        return encoder.encode(password);
    }

    // Método main para generar hashes de prueba
    public static void main(String[] args) {
        String[] passwords = {"password123", "admin", "usuario1"};
```

```
        System.out.println("=== Generador de Hashes BCrypt ===\n");
        for (String pwd : passwords) {
            String hash = generateHash(pwd);
            System.out.println("Password: " + pwd);
            System.out.println("Hash:     " + hash);
            System.out.println("Longitud: " + hash.length() + " caracteres\n");
        }
    }
}
```

5.3 Verificación de Contraseñas en el Login

```
/**
 * Proceso de verificación de login.
 */
public boolean verificarLogin(String username, String passwordIntroducida) {
    // 1. Buscar usuario por username
    Optional<Usuario> usuarioOpt = usuarioRepository.findByUsername(username);

    if (usuarioOpt.isEmpty()) {
        // Usuario no existe
        return false;
    }

    Usuario usuario = usuarioOpt.get();

    // 2. Verificar que el usuario esté activo
    if (!usuario.getActivo()) {
        // Usuario desactivado
        return false;
    }

    // 3. Verificar contraseña con BCrypt
    // NUNCA comparar strings directamente: passwordIntroducida.equals(usuario.getP
    return passwordUtil.verifyPassword(passwordIntroducida, usuario.getPasswordHash
}
```

6. Entidad JPA Completa con Todas las Tecnologías

Ahora combinamos todo lo aprendido en una entidad completa:

```java
package com.ejemplo.gestionusuarios.entity;

import jakarta.persistence.*;
import lombok.*;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import java.time.LocalDateTime;

/**
 * Entidad JPA que representa un usuario del sistema.
 *
 * Tecnologías utilizadas:
 * - JPA/Hibernate: Mapeo objeto-relacional
 * - Lombok: Reducción de código boilerplate
 * - BCrypt: Verificación segura de contraseñas
 */
@Entity
@Table(name = "usuarios", indexes = {
    @Index(name = "idx_username", columnList = "username"),
    @Index(name = "idx_email", columnList = "email")
})
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Usuario {

    // ========================================
    // CAMPOS DE LA ENTIDAD
    // ========================================

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

    @Column(name = "username", nullable = false, unique = true, length = 50)
    private String username;

    @Column(name = "email", nullable = false, unique = true, length = 100)
    private String email;

    @Column(name = "password_hash", nullable = false, length = 255)
    private String passwordHash;

    @Column(name = "activo", nullable = false)
    @Builder.Default  // Lombok: valor por defecto en el builder
    private Boolean activo = true;

    @Column(name = "fecha_creacion", updatable = false)
    private LocalDateTime fechaCreacion;

    @Column(name = "fecha_actualizacion")
    private LocalDateTime fechaActualizacion;

    // ========================================
    // CALLBACKS DEL CICLO DE VIDA
    // ========================================
```

```java
    /**
     * Se ejecuta automáticamente ANTES de insertar en BBDD.
     * Establece la fecha de creación y actualización.
     */
    @PrePersist
    protected void onCreate() {
        this.fechaCreacion = LocalDateTime.now();
        this.fechaActualizacion = LocalDateTime.now();
    }

    /**
     * Se ejecuta automáticamente ANTES de actualizar en BBDD.
     * Actualiza la fecha de última modificación.
     */
    @PreUpdate
    protected void onUpdate() {
        this.fechaActualizacion = LocalDateTime.now();
    }

    // ========================================
    // MÉTODOS DE NEGOCIO
    // ========================================

    // Encoder estático para verificación de contraseñas
    private static final BCryptPasswordEncoder encoder = new BCryptPasswordEncoder(

    /**
     * Verifica si la contraseña proporcionada coincide con el hash almacenado.
     *
     * @param rawPassword Contraseña en texto plano a verificar
     * @return true si la contraseña es correcta, false en caso contrario
     */
    public boolean verificarPassword(String rawPassword) {
        return encoder.matches(rawPassword, this.passwordHash);
    }

    /**
     * Hashea una contraseña y la asigna a este usuario.
     * Útil al crear o actualizar la contraseña.
     *
     * @param rawPassword Contraseña en texto plano
     */
    public void setPassword(String rawPassword) {
        this.passwordHash = encoder.encode(rawPassword);
    }

    /**
     * Desactiva el usuario (borrado lógico).
     */
    public void desactivar() {
        this.activo = false;
    }

    /**
     * Reactiva un usuario desactivado.
     */
    public void activar() {
        this.activo = true;
```

```
        }
    }
```

7. Servicio CRUD Completo

```java
package com.ejemplo.gestionusuarios.service;

import com.ejemplo.gestionusuarios.entity.Usuario;
import com.ejemplo.gestionusuarios.repository.UsuarioRepository;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.Optional;

/**
 * Servicio que implementa la lógica de negocio para gestión de usuarios.
 *
 * Anotaciones clave:
 * - @Service: Marca la clase como componente de servicio Spring
 * - @Transactional: Gestión automática de transacciones
 * - @Slf4j: Logger automático de Lombok
 * - @RequiredArgsConstructor: Inyección de dependencias por constructor
 */
@Service
@Transactional
@Slf4j
@RequiredArgsConstructor
public class UsuarioService {

    private final UsuarioRepository usuarioRepository;
    private final BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder

    // ========================================
    // OPERACIONES CREATE
    // ========================================

    /**
     * Crea un nuevo usuario con contraseña hasheada.
     *
     * @param username Nombre de usuario único
     * @param email Email único
     * @param rawPassword Contraseña en texto plano (se hasheará)
     * @return Usuario creado
     * @throws IllegalArgumentException si username o email ya existen
     */
    public Usuario crearUsuario(String username, String email, String rawPassword)
        log.info("Creando nuevo usuario: {}", username);

        // Validaciones de negocio
        if (usuarioRepository.existsByUsername(username)) {
            throw new IllegalArgumentException("El username '" + username + "' ya e
        }
        if (usuarioRepository.existsByEmail(email)) {
            throw new IllegalArgumentException("El email '" + email + "' ya está re
        }
```

```
        // Crear usuario con contraseña hasheada
        Usuario usuario = Usuario.builder()
                .username(username)
                .email(email)
                .passwordHash(passwordEncoder.encode(rawPassword))
                .activo(true)
                .build();

        Usuario guardado = usuarioRepository.save(usuario);
        log.info("Usuario creado con ID: {}", guardado.getId());

        return guardado;
    }

    // ========================================
    // OPERACIONES READ
    // ========================================

    /**
     * Obtiene todos los usuarios.
     */
    @Transactional(readOnly = true)
    public List<Usuario> obtenerTodos() {
        log.debug("Obteniendo todos los usuarios");
        return usuarioRepository.findAll();
    }

    /**
     * Obtiene un usuario por su ID.
     */
    @Transactional(readOnly = true)
    public Optional<Usuario> obtenerPorId(Long id) {
        log.debug("Buscando usuario con ID: {}", id);
        return usuarioRepository.findById(id);
    }

    /**
     * Obtiene un usuario por su username.
     */
    @Transactional(readOnly = true)
    public Optional<Usuario> obtenerPorUsername(String username) {
        log.debug("Buscando usuario con username: {}", username);
        return usuarioRepository.findByUsername(username);
    }

    /**
     * Obtiene todos los usuarios activos.
     */
    @Transactional(readOnly = true)
    public List<Usuario> obtenerUsuariosActivos() {
        return usuarioRepository.findByActivoTrue();
    }

    // ========================================
    // OPERACIONES UPDATE
    // ========================================

    /**
     * Actualiza email y/o contraseña de un usuario.
```

```
     *
     * @param id ID del usuario a actualizar
     * @param nuevoEmail Nuevo email (null para no cambiar)
     * @param nuevaPassword Nueva contraseña en texto plano (null para no cambiar)
     * @return Usuario actualizado
     * @throws IllegalArgumentException si el usuario no existe o email duplicado
     */
    public Usuario actualizarUsuario(Long id, String nuevoEmail, String nuevaPasswo
        log.info("Actualizando usuario ID: {}", id);

        Usuario usuario = usuarioRepository.findById(id)
                .orElseThrow(() -> new IllegalArgumentException("Usuario no encontr

        // Actualizar email si se proporciona
        if (nuevoEmail != null && !nuevoEmail.equals(usuario.getEmail())) {
            if (usuarioRepository.existsByEmail(nuevoEmail)) {
                throw new IllegalArgumentException("El email '" + nuevoEmail + "' y
            }
            usuario.setEmail(nuevoEmail);
            log.debug("Email actualizado a: {}", nuevoEmail);
        }

        // Actualizar contraseña si se proporciona
        if (nuevaPassword != null && !nuevaPassword.isEmpty()) {
            usuario.setPasswordHash(passwordEncoder.encode(nuevaPassword));
            log.debug("Contraseña actualizada para usuario: {}", usuario.getUsernam
        }

        return usuarioRepository.save(usuario);
    }

    // ========================================
    // OPERACIONES DELETE
    // ========================================

    /**
     * Elimina un usuario de forma permanente (borrado físico).
     *
     * @param id ID del usuario a eliminar
     * @throws IllegalArgumentException si el usuario no existe
     */
    public void eliminarUsuario(Long id) {
        log.warn("Eliminando usuario ID: {} (borrado físico)", id);

        if (!usuarioRepository.existsById(id)) {
            throw new IllegalArgumentException("Usuario no encontrado con ID: " + i
        }

        usuarioRepository.deleteById(id);
        log.info("Usuario eliminado permanentemente");
    }

    /**
     * Desactiva un usuario (borrado lógico).
     * El usuario permanece en la BBDD pero no puede hacer login.
     *
     * @param id ID del usuario a desactivar
     * @return Usuario desactivado
     */
```

```
    public Usuario desactivarUsuario(Long id) {
        log.info("Desactivando usuario ID: {} (borrado lógico)", id);

        Usuario usuario = usuarioRepository.findById(id)
                .orElseThrow(() -> new IllegalArgumentException("Usuario no encontr

        usuario.setActivo(false);
        return usuarioRepository.save(usuario);
    }

    /**
     * Reactiva un usuario previamente desactivado.
     */
    public Usuario activarUsuario(Long id) {
        log.info("Reactivando usuario ID: {}", id);

        Usuario usuario = usuarioRepository.findById(id)
                .orElseThrow(() -> new IllegalArgumentException("Usuario no encontr

        usuario.setActivo(true);
        return usuarioRepository.save(usuario);
    }

    // ========================================
    // OPERACIONES DE AUTENTICACIÓN
    // ========================================

    /**
     * Verifica las credenciales de un usuario para login.
     *
     * @param username Nombre de usuario
     * @param rawPassword Contraseña en texto plano
     * @return Optional con el usuario si las credenciales son válidas
     */
    @Transactional(readOnly = true)
    public Optional<Usuario> verificarCredenciales(String username, String rawPassw
        log.debug("Verificando credenciales para: {}", username);

        Optional<Usuario> usuarioOpt = usuarioRepository.findByUsername(username);

        if (usuarioOpt.isEmpty()) {
            log.warn("Intento de login con usuario inexistente: {}", username);
            return Optional.empty();
        }

        Usuario usuario = usuarioOpt.get();

        if (!usuario.getActivo()) {
            log.warn("Intento de login con usuario desactivado: {}", username);
            return Optional.empty();
        }

        if (passwordEncoder.matches(rawPassword, usuario.getPasswordHash())) {
            log.info("Login exitoso para usuario: {}", username);
            return Optional.of(usuario);
        }

        log.warn("Contraseña incorrecta para usuario: {}", username);
        return Optional.empty();
```

```
        }
    }
```

8. Clase Principal con Demostración

```
package com.ejemplo.gestionusuarios;

import com.ejemplo.gestionusuarios.entity.Usuario;
import com.ejemplo.gestionusuarios.service.UsuarioService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

import java.util.List;
import java.util.Optional;

@SpringBootApplication
@Slf4j
@RequiredArgsConstructor
public class GestionUsuariosApplication {

    public static void main(String[] args) {
        SpringApplication.run(GestionUsuariosApplication.class, args);
    }

    /**
     * CommandLineRunner que se ejecuta al iniciar la aplicación.
     * Demuestra todas las operaciones CRUD.
     */
    @Bean
    CommandLineRunner demoCRUD(UsuarioService usuarioService) {
        return args -> {
            System.out.println("\n" + "=".repeat(60));
            System.out.println("   DEMOSTRACIÓN CRUD - Sistema de Gestión de Usuari
            System.out.println("=".repeat(60) + "\n");

            // ==========================================
            // 1. TEST DE CONEXIÓN
            // ==========================================
            System.out.println("1. TEST DE CONEXIÓN A BASE DE DATOS");
            System.out.println("-".repeat(40));
            try {
                List<Usuario> usuarios = usuarioService.obtenerTodos();
                System.out.println("✅ Conexión exitosa!");
                System.out.println("   Usuarios en BBDD: " + usuarios.size());
            } catch (Exception e) {
                System.out.println("❌ Error de conexión: " + e.getMessage());
                return;
            }
            System.out.println();

            // ==========================================
            // 2. CREATE - Crear nuevo usuario
            // ==========================================
            System.out.println("2. CREATE - Crear nuevo usuario");
            System.out.println("-".repeat(40));
            Usuario nuevoUsuario = null;
```

```
try {
    nuevoUsuario = usuarioService.crearUsuario(
        "test_user",
        "test@ejemplo.com",
        "password123"
    );
    System.out.println("✅ Usuario creado:");
    System.out.println("   ID: " + nuevoUsuario.getId());
    System.out.println("   Username: " + nuevoUsuario.getUsername());
    System.out.println("   Email: " + nuevoUsuario.getEmail());
    System.out.println("   Hash (primeros 20 chars): " +
        nuevoUsuario.getPasswordHash().substring(0, 20) + "...");
} catch (IllegalArgumentException e) {
    System.out.println("⚠️ " + e.getMessage());
    // Si ya existe, lo buscamos
    nuevoUsuario = usuarioService.obtenerPorUsername("test_user").orEls
}
System.out.println();

// ============================================
// 3. READ - Listar todos los usuarios
// ============================================
System.out.println("3. READ - Listar todos los usuarios");
System.out.println("-".repeat(40));
List<Usuario> todosUsuarios = usuarioService.obtenerTodos();
System.out.printf("%-5s %-15s %-25s %-8s%n", "ID", "USERNAME", "EMAIL",
System.out.println("-".repeat(55));
for (Usuario u : todosUsuarios) {
    System.out.printf("%-5d %-15s %-25s %-8s%n",
        u.getId(),
        u.getUsername(),
        u.getEmail().length() > 24 ? u.getEmail().substring(0, 22) + ".
        u.getActivo() ? "Sí" : "No"
    );
}
System.out.println();

// ============================================
// 4. READ - Buscar por username
// ============================================
System.out.println("4. READ - Buscar usuario por username");
System.out.println("-".repeat(40));
Optional<Usuario> encontrado = usuarioService.obtenerPorUsername("admin
if (encontrado.isPresent()) {
    Usuario u = encontrado.get();
    System.out.println("✅ Usuario 'admin' encontrado:");
    System.out.println("   ID: " + u.getId());
    System.out.println("   Email: " + u.getEmail());
    System.out.println("   Creado: " + u.getFechaCreacion());
} else {
    System.out.println("❌ Usuario 'admin' no encontrado");
}
System.out.println();

// ============================================
// 5. UPDATE - Actualizar email
// ============================================
System.out.println("5. UPDATE - Actualizar email de usuario");
System.out.println("-".repeat(40));
```

```
if (nuevoUsuario != null) {
    Usuario actualizado = usuarioService.actualizarUsuario(
        nuevoUsuario.getId(),
        "nuevo_email@ejemplo.com",
        null  // No cambiar contraseña
    );
    System.out.println("✅ Email actualizado:");
    System.out.println("   Email anterior: test@ejemplo.com");
    System.out.println("   Email nuevo: " + actualizado.getEmail());
    System.out.println("   Fecha actualización: " + actualizado.getFech
}
System.out.println();

// ============================================
// 6. VERIFICAR CREDENCIALES (Login)
// ============================================
System.out.println("6. VERIFICAR CREDENCIALES - Simulación de login");
System.out.println("-".repeat(40));

// Login correcto
Optional<Usuario> loginOk = usuarioService.verificarCredenciales("test_
System.out.println("Login (test_user/password123): " +
    (loginOk.isPresent() ? "✅ ÉXITO" : "❌ FALLIDO"));

// Login incorrecto
Optional<Usuario> loginFail = usuarioService.verificarCredenciales("tes
System.out.println("Login (test_user/wrongpass): " +
    (loginFail.isPresent() ? "✅ ÉXITO" : "❌ FALLIDO (esperado)"));
System.out.println();

// ============================================
// 7. DELETE LÓGICO - Desactivar usuario
// ============================================
System.out.println("7. DELETE LÓGICO - Desactivar usuario");
System.out.println("-".repeat(40));
if (nuevoUsuario != null) {
    Usuario desactivado = usuarioService.desactivarUsuario(nuevoUsuario
    System.out.println("✅ Usuario desactivado:");
    System.out.println("   Username: " + desactivado.getUsername());
    System.out.println("   Activo: " + desactivado.getActivo());

    // Intentar login con usuario desactivado
    Optional<Usuario> loginDesactivado =
        usuarioService.verificarCredenciales("test_user", "password123"
    System.out.println("   Login tras desactivar: " +
        (loginDesactivado.isPresent() ? "✅ ÉXITO" : "❌ BLOQUEADO (es
}
System.out.println();

// ============================================
// 8. DELETE FÍSICO - Eliminar usuario
// ============================================
System.out.println("8. DELETE FÍSICO - Eliminar usuario permanentemente
System.out.println("-".repeat(40));
if (nuevoUsuario != null) {
    Long idEliminar = nuevoUsuario.getId();
    usuarioService.eliminarUsuario(idEliminar);
    System.out.println("✅ Usuario eliminado permanentemente");
```

```
                // Verificar eliminación
                Optional<Usuario> verificar = usuarioService.obtenerPorId(idElimina
                System.out.println("   Buscar ID " + idEliminar + ": " +
                    (verificar.isPresent() ? "Encontrado" : "No encontrado (correct
            }
            System.out.println();

            // ========================================
            // RESUMEN FINAL
            // ========================================
            System.out.println("=".repeat(60));
            System.out.println("   DEMOSTRACIÓN COMPLETADA EXITOSAMENTE");
            System.out.println("=".repeat(60));
            System.out.println("\nUsuarios finales en BBDD: " +
                usuarioService.obtenerTodos().size());
        };
    }
}
```

9. Script SQL para Crear la Base de Datos

```sql
-- ========================================
-- Script: 01_crear_base_datos.sql
-- Sistema de Gestión de Usuarios
-- ========================================

-- Crear la base de datos
CREATE DATABASE IF NOT EXISTS gestion_usuarios
    CHARACTER SET utf8mb4
    COLLATE utf8mb4_unicode_ci;

USE gestion_usuarios;

-- Crear la tabla de usuarios
CREATE TABLE IF NOT EXISTS usuarios (
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) NOT NULL UNIQUE,
    email VARCHAR(100) NOT NULL UNIQUE,
    password_hash VARCHAR(255) NOT NULL,
    activo BOOLEAN NOT NULL DEFAULT TRUE,
    fecha_creacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    fecha_actualizacion TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMES

    -- Índices para optimizar búsquedas
    INDEX idx_username (username),
    INDEX idx_email (email),
    INDEX idx_activo (activo)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;

-- ========================================
-- Insertar usuarios de prueba
-- Contraseñas: todas son "password123"
-- Hash generado con BCrypt strength=12
-- ========================================

INSERT INTO usuarios (username, email, password_hash, activo) VALUES
('admin', 'admin@empresa.com',
 '$2a$12$LQv3c1yqBWVHxkd0LHAkCOYz6TtxMQJqhN8/X4.IZPhFD3F4Wvqay', TRUE),

('usuario1', 'usuario1@empresa.com',
 '$2a$12$LQv3c1yqBWVHxkd0LHAkCOYz6TtxMQJqhN8/X4.IZPhFD3F4Wvqay', TRUE),

('usuario2', 'usuario2@empresa.com',
 '$2a$12$LQv3c1yqBWVHxkd0LHAkCOYz6TtxMQJqhN8/X4.IZPhFD3F4Wvqay', TRUE),

('inactivo', 'inactivo@empresa.com',
 '$2a$12$LQv3c1yqBWVHxkd0LHAkCOYz6TtxMQJqhN8/X4.IZPhFD3F4Wvqay', FALSE);

-- Verificar inserción
SELECT id, username, email, activo, fecha_creacion FROM usuarios;
```

10. Resumen de Tecnologías

Tabla 2 Resumen del Stack Tecnológico

Tecnología	Función	Anotaciones/Clases Clave
Spring Boot	Framework principal	`@SpringBootApplication`, `CommandLineRunner`
Spring Data JPA	Repositorios y CRUD	`JpaRepository`, `@Query`, `@Repository`
Hibernate	ORM (Mapeo O-R)	`@Entity`, `@Table`, `@Column`, `@Id`
JPA Lifecycle	Eventos del ciclo de vida	`@PrePersist`, `@PreUpdate`
Transacciones	Gestión ACID	`@Transactional`, `@Transactional(readOnly=true)`
BCrypt	Seguridad contraseñas	`BCryptPasswordEncoder`, `encode()`, `matches()`
Lombok	Reducción código	`@Data`, `@Builder`, `@Slf4j`, `@RequiredArgsConstructor`
MySQL	Base de datos	`mysql-connector-j`, `HikariCP`

Siguiente paso

Ahora que conoces las tecnologías, practica implementando el sistema CRUD completo siguiendo la estructura del proyecto. Recuerda ejecutar el script SQL antes de arrancar la aplicación.

11. Ejercicios Propuestos

Ejercicio 1: Añadir campo «último login»

Añade un campo `ultimoLogin` (LocalDateTime) a la entidad Usuario y actualízalo cada vez que se verifiquen las credenciales correctamente.

Ejercicio 2: Implementar cambio de contraseña

Crea un método `cambiarPassword(Long id, String passwordActual, String passwordNueva)` que verifique la contraseña actual antes de permitir el cambio.

Ejercicio 3: Búsqueda paginada

Implementa un método en el repositorio que devuelva usuarios paginados usando `Pageable` de Spring Data.

Ejercicio 4: Contador de intentos fallidos

Añade un campo `intentosFallidos` y bloquea automáticamente al usuario después de 3 intentos fallidos.

Soluciones