package com.example.inventory\_management.service.impl;

import com.example.inventory\_management.entity.Category;

import com.example.inventory\_management.entity.Product;

import com.example.inventory\_management.repository.ProductRepository;

import com.example.inventory\_management.service.ProductService;

import org.springframework.stereotype.Service;

import java.time.LocalDate;

import java.util.\*;

@Service

public class ProductServiceImpl implements ProductService {

private final ProductRepository productRepository;

private static final double DISCOUNT\_RATE = 0.2;

public ProductServiceImpl(ProductRepository productRepository) {

this.productRepository = productRepository;

}

@Override

public void addProduct(Product product) {

productRepository.save(product);

}

@Override

public void addProductsBulk(List<Product> products) {

for (var product : products) {

addProduct(product);

}

}

@Override

public void updateProduct(Product product) {

if (productRepository.findById(product.getId()) == null) return;

productRepository.update(product);

}

@Override

public void deleteProduct(Long id) {

productRepository.deleteById(id);

}

@Override

public Product findById(Long id) {

return productRepository.findById(id);

}

@Override

public List<Product> getExpiringProducts() {

LocalDate today = LocalDate.now();

LocalDate next7days = today.plusDays(7);

List<Product> expiring = new ArrayList<>();

for (Product p : productRepository.findAll()) {

if (p.getExpiryDate() == null) {

p.setDiscount(null);

p.setAvailable(true);

continue;

}

if (p.getExpiryDate().isBefore(today)) {

p.setAvailable(false);

p.setDiscount(null);

} else if (!p.getExpiryDate().isAfter(next7days)) {

p.setDiscount(DISCOUNT\_RATE);

p.setAvailable(true);

expiring.add(p);

} else {

p.setDiscount(null);

p.setAvailable(true);

}

}

return expiring;

}

@Override

public List<Product> getAllProducts() {

getExpiringProducts();

return productRepository.findAll();

}

@Override

public Map<Category, Double> getTotalValueByCategory() {

Map<Category, Double> map = new HashMap<>();

for (var p : productRepository.findAll()) {

if (p.isAvailable()) {

map.merge(p.getCategory(), p.getPrice() \* p.getQuantity(), Double::sum);

}

}

return map;

}

@Override

public Map<Category, List<Product>> getProductsByCategoryWithDiscounts() {

Map<Category, List<Product>> map = new HashMap<>();

for (var p : productRepository.findAll()) {

map.computeIfAbsent(p.getCategory(), k -> new ArrayList<>()).add(p);

}

return map;

}

}

package com.example.inventory\_management.service;

import com.example.inventory\_management.entity.Category;

import com.example.inventory\_management.entity.Product;

import java.util.List;

import java.util.Map;

public interface ProductService {

void addProduct(Product product);

void addProductsBulk(List<Product> products);

void updateProduct(Product product);

void deleteProduct(Long id);

Product findById(Long id);

List<Product> getExpiringProducts();

List<Product> getAllProducts();

Map<Category, Double> getTotalValueByCategory();

Map<Category, List<Product>> getProductsByCategoryWithDiscounts();

}

package com.example.inventory\_management.aspect;

import jakarta.servlet.http.HttpServletRequest;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.\*;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.stereotype.Component;

import java.time.LocalDateTime;

@Aspect

@Component

public class LoggingAspect {

private final Logger logger = LoggerFactory.getLogger("applicationLogger");

@Pointcut("within(com.example.inventory\_management.controller..\*)")

public void controllerLayer() {}

@Before("controllerLayer()")

public void logBefore(JoinPoint joinPoint) {

Authentication auth = SecurityContextHolder.getContext().getAuthentication();

String username = (auth != null) ? auth.getName() : "Anonymous";

logger.info("User '{}' accessed method '{}' at {}", username, joinPoint.getSignature().toShortString(), LocalDateTime.now());

}

}

package com.example.inventory\_management.util;

import org.springframework.mail.SimpleMailMessage;

import org.springframework.mail.javamail.JavaMailSender;

import org.springframework.stereotype.Service;

@Service

public class MailService {

private final JavaMailSender mailSender;

public MailService(JavaMailSender sender) {

this.mailSender = sender;

}

public void sendSimpleMail(String to, String subject, String body) {

SimpleMailMessage msg = new SimpleMailMessage();

msg.setTo(to);

msg.setSubject(subject);

msg.setText(body);

mailSender.send(msg);

}

}

package com.example.inventory\_management.repository;

import com.example.inventory\_management.entity.Category;

import com.example.inventory\_management.entity.Product;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.core.RowMapper;

import org.springframework.stereotype.Repository;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.time.LocalDate;

import java.util.List;

@Repository

public class ProductRepository {

private final JdbcTemplate jdbcTemplate;

private final RowMapper<Product> rowMapper = new RowMapper<>() {

@Override

public Product mapRow(ResultSet rs, int rowNum) throws SQLException {

Product p = new Product();

p.setId(rs.getLong("id"));

p.setName(rs.getString("name"));

p.setCategory(Category.valueOf(rs.getString("category")));

p.setPrice(rs.getDouble("price"));

p.setQuantity(rs.getInt("quantity"));

p.setExpiryDate(rs.getDate("expiry\_date") != null ? rs.getDate("expiry\_date").toLocalDate() : null);

p.setDiscount(rs.getObject("discount", Double.class));

p.setAvailable(rs.getBoolean("available"));

return p;

}

};

public ProductRepository(JdbcTemplate jdbcTemplate) {

this.jdbcTemplate = jdbcTemplate;

}

public List<Product> findAll() {

return jdbcTemplate.query("SELECT \* FROM products", rowMapper);

}

public Product findById(Long id) {

List<Product> products = jdbcTemplate.query("SELECT \* FROM products WHERE id = ?", rowMapper, id);

return products.isEmpty() ? null : products.get(0);

}

public int save(Product product) {

return jdbcTemplate.update("INSERT INTO products (name, category, price, quantity, expiry\_date, discount, available) VALUES (?, ?, ?, ?, ?, ?, ?)",

product.getName(),

product.getCategory().name(),

product.getPrice(),

product.getQuantity(),

product.getExpiryDate(),

product.getDiscount(),

product.isAvailable());

}

public int update(Product product) {

return jdbcTemplate.update("UPDATE products SET name = ?, category = ?, price = ?, quantity = ?, expiry\_date = ?, discount = ?, available = ? WHERE id = ?",

product.getName(),

product.getCategory().name(),

product.getPrice(),

product.getQuantity(),

product.getExpiryDate(),

product.getDiscount(),

product.isAvailable(),

product.getId());

}

public int deleteById(Long id) {

return jdbcTemplate.update("DELETE FROM products WHERE id = ?", id);

}

}

package com.example.inventory\_management.dto;

public class RegistrationRequest {

private String username;

private String email;

private String password;

// getters/setters

}

package com.example.inventory\_management.dto;

public class LoginRequest {

private String username;

private String password;

// getters/setters

}

package com.example.inventory\_management.service.impl;

import com.example.inventory\_management.dto.LoginRequest;

import com.example.inventory\_management.dto.RegistrationRequest;

import com.example.inventory\_management.entity.User;

import com.example.inventory\_management.repository.UserRepository;

import com.example.inventory\_management.service.AuthService;

import com.example.inventory\_management.config.JwtUtils;

import com.example.inventory\_management.util.MailService;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.stereotype.Service;

@Service

public class AuthServiceImpl implements AuthService {

private final UserRepository userRepository;

private final PasswordEncoder passwordEncoder;

private final JwtUtils jwtUtils;

private final MailService mailService;

public AuthServiceImpl(UserRepository userRepository, PasswordEncoder passwordEncoder, JwtUtils jwtUtils, MailService mailService) {

this.userRepository = userRepository;

this.passwordEncoder = passwordEncoder;

this.jwtUtils = jwtUtils;

this.mailService = mailService;

}

@Override

public void register(RegistrationRequest request) {

if (userRepository.findByUsername(request.getUsername()) != null) {

throw new RuntimeException("Username already exists");

}

if (userRepository.findByEmail(request.getEmail()) != null) {

throw new RuntimeException("Email already exists");

}

User user = new User();

user.setUsername(request.getUsername());

user.setEmail(request.getEmail());

user.setPassword(passwordEncoder.encode(request.getPassword()));

user.setRole("USER");

user.setEnabled(true);

userRepository.save(user);

// Optionally send welcome email here

mailService.sendSimpleMail(user.getEmail(), "Welcome to Inventory Management", "Thanks for registering!");

}

@Override

public String login(LoginRequest request) {

User user = userRepository.findByUsername(request.getUsername());

if (user == null || !passwordEncoder.matches(request.getPassword(), user.getPassword())) {

throw new RuntimeException("Invalid username or password");

}

return jwtUtils.generateToken(user.getUsername(), user.getRole());

}

// Other methods (changePassword, forgotPassword, updateProfile) will come next

}

@Override

public void changePassword(String username, String oldPassword, String newPassword) {

User user = userRepository.findByUsername(username);

if (user == null) throw new RuntimeException("User not found");

if (!passwordEncoder.matches(oldPassword, user.getPassword())) {

throw new RuntimeException("Old password incorrect");

}

userRepository.updatePassword(user.getId(), passwordEncoder.encode(newPassword));

}

@Override

public void forgotPassword(String email) {

User user = userRepository.findByEmail(email);

if (user == null) throw new RuntimeException("Email not found");

// Generate reset token (simple random string here, improve in prod)

String resetToken = java.util.UUID.randomUUID().toString();

// Send email with reset link (you can implement a reset URL in frontend)

String message = "To reset your password, use this token: " + resetToken;

mailService.sendSimpleMail(email, "Password Reset", message);

// You should save this token with expiry in DB in a real app (omitted for brevity)

}

@Override

public void updateProfile(String username, String newUsername, String newEmail) {

User user = userRepository.findByUsername(username);

if (user == null) throw new RuntimeException("User not found");

userRepository.updateProfile(user.getId(), newUsername, newEmail);

}

package com.example.inventory\_management.service;

import com.example.inventory\_management.dto.LoginRequest;

import com.example.inventory\_management.dto.RegistrationRequest;

public interface AuthService {

void register(RegistrationRequest request);

String login(LoginRequest request);

void changePassword(String username, String oldPassword, String newPassword);

void forgotPassword(String email);

void updateProfile(String username, String newUsername, String newEmail);

}

package com.example.inventory\_management.service.impl;

import com.example.inventory\_management.entity.User;

import com.example.inventory\_management.repository.UserRepository;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

import org.springframework.security.core.userdetails.\*;

import org.springframework.stereotype.Service;

import java.util.Collections;

@Service

public class CustomUserDetailsService implements UserDetailsService {

private final UserRepository userRepository;

public CustomUserDetailsService(UserRepository repo) {

this.userRepository = repo;

}

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

User user = userRepository.findByUsername(username);

if (user == null) throw new UsernameNotFoundException("User not found");

return new org.springframework.security.core.userdetails.User(

user.getUsername(),

user.getPassword(),

user.isEnabled(),

true,

true,

true,

Collections.singleton(new SimpleGrantedAuthority("ROLE\_" + user.getRole()))

);

}

}

package com.example.inventory\_management.repository;

import com.example.inventory\_management.entity.User;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.core.RowMapper;

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public class UserRepository {

private final JdbcTemplate jdbcTemplate;

private final RowMapper<User> userRowMapper = (rs, rowNum) -> {

User u = new User();

u.setId(rs.getLong("id"));

u.setUsername(rs.getString("username"));

u.setEmail(rs.getString("email"));

u.setPassword(rs.getString("password"));

u.setRole(rs.getString("role"));

u.setEnabled(rs.getBoolean("enabled"));

return u;

};

public UserRepository(JdbcTemplate jdbcTemplate) {

this.jdbcTemplate = jdbcTemplate;

}

public User findByUsername(String username) {

List<User> users = jdbcTemplate.query(

"SELECT \* FROM users WHERE username = ?",

userRowMapper,

username);

return users.isEmpty() ? null : users.get(0);

}

public User findByEmail(String email) {

List<User> users = jdbcTemplate.query(

"SELECT \* FROM users WHERE email = ?",

userRowMapper,

email);

return users.isEmpty() ? null : users.get(0);

}

public int save(User user) {

return jdbcTemplate.update(

"INSERT INTO users (username, email, password, role, enabled) VALUES (?, ?, ?, ?, ?)",

user.getUsername(),

user.getEmail(),

user.getPassword(),

user.getRole(),

user.isEnabled()

);

}

public int updatePassword(Long id, String newPassword) {

return jdbcTemplate.update(

"UPDATE users SET password = ? WHERE id = ?",

newPassword,

id

);

}

public int updateProfile(Long id, String username, String email) {

return jdbcTemplate.update(

"UPDATE users SET username = ?, email = ? WHERE id = ?",

username,

email,

id

);

}

public User findById(Long id) {

List<User> users = jdbcTemplate.query(

"SELECT \* FROM users WHERE id = ?",

userRowMapper,

id);

return users.isEmpty() ? null : users.get(0);

}

}

package com.example.inventory\_management.entity;

public class User {

private Long id;

private String username;

private String email;

private String password; // hashed

private String role; // "ADMIN" or "USER"

private boolean enabled;

public User() {}

public User(Long id, String username, String email, String password, String role, boolean enabled) {

this.id = id;

this.username = username;

this.email = email;

this.password = password;

this.role = role;

this.enabled = enabled;

}

// Getters and setters

// ...

}

package com.example.inventory\_management.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.cors.CorsConfiguration;

import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

import org.springframework.web.filter.CorsFilter;

import java.util.List;

@Configuration

public class CorsConfig {

@Bean

public CorsFilter corsFilter() {

var config = new CorsConfiguration();

config.setAllowedOrigins(List.of("http://localhost:3000")); // Adjust frontend origin

config.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE", "OPTIONS"));

config.setAllowedHeaders(List.of("\*"));

config.setAllowCredentials(true);

var source = new UrlBasedCorsConfigurationSource();

source.registerCorsConfiguration("/\*\*", config);

return new CorsFilter(source);

}

}

package com.example.inventory\_management.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.http.HttpMethod;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.AuthenticationProvider;

import org.springframework.security.authentication.dao.DaoAuthenticationProvider;

import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.core.userdetails.UserDetailsService;

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

import org.springframework.security.web.SecurityFilterChain;

import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;

@Configuration

public class SecurityConfig {

private final JwtAuthenticationFilter jwtAuthenticationFilter;

private final UserDetailsService userDetailsService;

public SecurityConfig(JwtAuthenticationFilter jwtAuthenticationFilter,

UserDetailsService userDetailsService) {

this.jwtAuthenticationFilter = jwtAuthenticationFilter;

this.userDetailsService = userDetailsService;

}

@Bean

public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {

http

.csrf(csrf -> csrf.disable())

.cors(cors -> cors.disable()) // We'll configure CORS properly below

.authorizeHttpRequests(auth -> auth

.requestMatchers(HttpMethod.POST, "/auth/\*\*").permitAll()

.requestMatchers("/admin/\*\*").hasRole("ADMIN")

.anyRequest().authenticated()

)

.sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))

.authenticationProvider(authenticationProvider())

.addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class);

return http.build();

}

@Bean

public AuthenticationProvider authenticationProvider() {

var provider = new DaoAuthenticationProvider();

provider.setUserDetailsService(userDetailsService);

provider.setPasswordEncoder(passwordEncoder());

return provider;

}

@Bean

public BCryptPasswordEncoder passwordEncoder() {

return new BCryptPasswordEncoder();

}

@Bean

public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {

return config.getAuthenticationManager();

}

}

package com.example.inventory\_management.config;

import jakarta.servlet.FilterChain;

import jakarta.servlet.ServletException;

import jakarta.servlet.http.HttpServletRequest;

import jakarta.servlet.http.HttpServletResponse;

import org.springframework.http.HttpHeaders;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.core.authority.SimpleGrantedAuthority;

import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;

import org.springframework.stereotype.Component;

import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

import java.util.List;

@Component

public class JwtAuthenticationFilter extends OncePerRequestFilter {

private final JwtUtils jwtUtils;

public JwtAuthenticationFilter(JwtUtils jwtUtils) {

this.jwtUtils = jwtUtils;

}

@Override

protected void doFilterInternal(HttpServletRequest request,

HttpServletResponse response,

FilterChain filterChain)

throws ServletException, IOException {

final String header = request.getHeader(HttpHeaders.AUTHORIZATION);

if (header == null || !header.startsWith("Bearer ")) {

filterChain.doFilter(request, response);

return;

}

final String token = header.substring(7);

if (!jwtUtils.validateToken(token)) {

filterChain.doFilter(request, response);

return;

}

String username = jwtUtils.getUsernameFromToken(token);

String role = jwtUtils.getRoleFromToken(token);

var authToken = new UsernamePasswordAuthenticationToken(

username,

null,

List.of(new SimpleGrantedAuthority("ROLE\_" + role))

);

authToken.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

SecurityContextHolder.getContext().setAuthentication(authToken);

filterChain.doFilter(request, response);

}

}

In application properties

jwt.secret=VerySecretJwtKeyChangeMe

jwt.expiration-ms=86400000 # 1 day in milliseconds

server.port=8081

spring.datasource.url=jdbc:mysql://localhost:3306/inventory\_db?useSSL=false&serverTimezone=UTC

spring.datasource.username=root

spring.datasource.password=YOUR\_DB\_PASSWORD

spring.mail.host=smtp.example.com

spring.mail.port=587

spring.mail.username=your-email@example.com

spring.mail.password=your-email-password

spring.mail.properties.mail.smtp.auth=true

spring.mail.properties.mail.smtp.starttls.enable=true

logging.file.name=application.log

package com.example.inventory\_management.config;

import com.auth0.jwt.JWT;

import com.auth0.jwt.algorithms.Algorithm;

import com.auth0.jwt.interfaces.DecodedJWT;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.stereotype.Component;

import java.util.Date;

@Component

public class JwtUtils {

@Value("${jwt.secret}")

private String jwtSecret;

@Value("${jwt.expiration-ms}")

private long jwtExpirationMs;

public String generateToken(String username, String role) {

return JWT.create()

.withSubject(username)

.withClaim("role", role)

.withExpiresAt(new Date(System.currentTimeMillis() + jwtExpirationMs))

.sign(Algorithm.HMAC256(jwtSecret));

}

public String getUsernameFromToken(String token) {

DecodedJWT decodedJWT = JWT.require(Algorithm.HMAC256(jwtSecret))

.build()

.verify(token);

return decodedJWT.getSubject();

}

public String getRoleFromToken(String token) {

DecodedJWT decodedJWT = JWT.require(Algorithm.HMAC256(jwtSecret))

.build()

.verify(token);

return decodedJWT.getClaim("role").asString();

}

public boolean validateToken(String token) {

try {

JWT.require(Algorithm.HMAC256(jwtSecret)).build().verify(token);

return true;

} catch (Exception e) {

return false;

}

}

}

Inventory\_management/

├── src/

│ ├── main/

│ │ ├── java/

│ │ │ └── com/example/inventory\_management/

│ │ │ ├── InventoryManagementApplication.java

│ │ │ ├── config/

│ │ │ │ ├── SecurityConfig.java

│ │ │ │ ├── JwtAuthenticationFilter.java

│ │ │ │ └── JwtUtils.java

│ │ │ ├── controller/

│ │ │ │ ├── AuthController.java

│ │ │ │ └── ProductController.java

│ │ │ ├── dto/

│ │ │ │ ├── LoginRequest.java

│ │ │ │ ├── RegistrationRequest.java

│ │ │ │ └── ...

│ │ │ ├── entity/

│ │ │ │ ├── User.java

│ │ │ │ ├── Product.java

│ │ │ │ └── Category.java

│ │ │ ├── repository/

│ │ │ │ ├── UserRepository.java

│ │ │ │ └── ProductRepository.java

│ │ │ ├── service/

│ │ │ │ ├── AuthService.java

│ │ │ │ ├── ProductService.java

│ │ │ │ ├── impl/

│ │ │ │ │ ├── AuthServiceImpl.java

│ │ │ │ │ └── ProductServiceImpl.java

│ │ │ ├── aspect/

│ │ │ │ └── LoggingAspect.java

│ │ │ └── util/

│ │ │ └── MailService.java

│ │ └── resources/

│ │ ├── application.properties

│ │ ├── schema.sql

│ │ └── data.sql

├── pom.xml

├── README.md

<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

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>Inventory\_management</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>Inventory\_management</name>

<description>Inventory management system with Spring Boot, JWT, Spring JDBC</description>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.1.0</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<properties>

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

</properties>

<dependencies>

<!-- Spring Boot Starter Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Boot Starter Security -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-security</artifactId>

</dependency>

<!-- Spring Boot Starter JDBC -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

<!-- MySQL Connector/J -->

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Java JWT -->

<dependency>

<groupId>com.auth0</groupId>

<artifactId>java-jwt</artifactId>

<version>4.2.1</version>

</dependency>

<!-- Spring Boot Starter AOP -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-aop</artifactId>

</dependency>

<!-- Java Mail Sender -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-mail</artifactId>

</dependency>

<!-- Lombok (optional for cleaner code, can skip if you want) -->

<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>

<!-- Spring Boot Maven Plugin -->

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

CREATE TABLE users (

id BIGINT AUTO\_INCREMENT PRIMARY KEY,

username VARCHAR(50) UNIQUE NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

password VARCHAR(255) NOT NULL,

role VARCHAR(20) NOT NULL,

enabled BOOLEAN NOT NULL

);

CREATE TABLE products (

id BIGINT AUTO\_INCREMENT PRIMARY KEY,

name VARCHAR(100) NOT NULL,

category VARCHAR(50),

price DOUBLE NOT NULL,

quantity INT NOT NULL,

expiry\_date DATE NULL,

discount DOUBLE NULL,

available BOOLEAN NOT NULL

);