File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\pom.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>3.4.5</version>
■■<relativePath/>
■</parent>

■<groupId>com.molsys</groupId>
■<artifactId>example2</artifactId>
■<version>0.0.1-SNAPSHOT</version>
■<name>example2</name>
■<description>Demo project for Spring Boot</description>

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

■<dependencies>
■■<!-- Spring Boot Starters -->
■■<dependency>
■■■<groupId>org.springframework.boot</groupId>
■■■<artifactId>spring-boot-starter-data-jpa</artifactId>
■■</dependency>
■■<dependency>
■■■<groupId>org.springframework.boot</groupId>
■■■<artifactId>spring-boot-starter-web</artifactId>
■■</dependency>
■■<dependency>
■■■<groupId>org.springframework.boot</groupId>
■■■<artifactId>spring-boot-starter-security</artifactId>
■■</dependency>
■■<dependency>
■■■<groupId>org.springframework.boot</groupId>
■■■<artifactId>spring-boot-starter-mail</artifactId>
■■</dependency>

■■<!-- JWT -->
■■<dependency>
■■■<groupId>io.jsonwebtoken</groupId>
■■■<artifactId>jjwt-api</artifactId>
■■■<version>0.11.5</version>
■■</dependency>
■■<dependency>
■■■<groupId>io.jsonwebtoken</groupId>
■■■<artifactId>jjwt-impl</artifactId>
■■■<version>0.11.5</version>
■■■<scope>runtime</scope>
■■</dependency>
■■<dependency>
■■■<groupId>io.jsonwebtoken</groupId>
■■■<artifactId>jjwt-jackson</artifactId>
■■■<version>0.11.5</version>
■■■<scope>runtime</scope>
■■</dependency>

■■<!-- MySQL -->
■■<dependency>
■■■<groupId>com.mysql</groupId>
■■■<artifactId>mysql-connector-j</artifactId>
■■■<version>8.3.0</version>
■■</dependency>

■■<!-- Lombok -->
■■<dependency>
■■■<groupId>org.projectlombok</groupId>
```

```
■■■<artifactId>lombok</artifactId>
■■■<optional>true</optional>
■■</dependency>

■■<!-- ? Spring Boot Test -->
■■<dependency>
■■■<groupId>org.springframework.boot</groupId>
■■■<artifactId>spring-boot-starter-test</artifactId>
■■■<scope>test</scope>
■■</dependency>

■■<dependency>
■■■<groupId>org.junit.jupiter</groupId>
■■■<artifactId>junit-jupiter</artifactId>
■■■<version>5.10.0</version>
■■■<scope>test</scope>
■■</dependency>

■■<!-- ? Mockito Core -->
■■<dependency>
■■■<groupId>org.mockito</groupId>
■■■<artifactId>mockito-core</artifactId>
■■■<version>5.10.0</version>
■■■<scope>test</scope>
■■</dependency>

■■<!-- ? Mockito JUnit Integration -->
■■<dependency>
■■■<groupId>org.mockito</groupId>
■■■<artifactId>mockito-junit-jupiter</artifactId>
■■■<version>5.10.0</version>
■■■<scope>test</scope>
■■</dependency>
■</dependencies>

■<build>
■■<plugins>
■■■<plugin>
■■■■<groupId>org.apache.maven.plugins</groupId>
■■■■<artifactId>maven-compiler-plugin</artifactId>
■■■■<configuration>
■■■■■<annotationProcessorPaths>
■■■■■■<path>
■■■■■■■■<groupId>org.projectlombok</groupId>
■■■■■■■■<artifactId>lombok</artifactId>
■■■■■■</path>
■■■■■</annotationProcessorPaths>
■■■■</configuration>
■■■</plugin>
■■■<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>
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\Example2Application.java

```
package com.molsys.example2;

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

@SpringBootApplication
```

```
public class  Example2Application {

■public static void main(String[] args) {
■■SpringApplication.run(Example2Application.class, args);
■}

}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\config\MailConfig.java

```
package com.molsys.example2.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.mail.javamail.JavaMailSenderImpl;
import org.springframework.beans.factory.annotation.Value;

import java.util.Properties;

@Configuration
public class MailConfig {

    @Value("${spring.mail.host}")
    private String host;

    @Value("${spring.mail.port}")
    private int port;

    @Value("${spring.mail.username}")
    private String username;

    @Value("${spring.mail.password}")
    private String password;

    @Value("${spring.mail.properties.mail.smtp.auth}")
    private boolean auth;

    @Value("${spring.mail.properties.mail.smtp.starttls.enable}")
    private boolean starttls;

    @Bean
    public JavaMailSender javaMailSender() {
        JavaMailSenderImpl mailSender = new JavaMailSenderImpl();
        mailSender.setHost(host);
        mailSender.setPort(port);
        mailSender.setUsername(username);
        mailSender.setPassword(password);

        Properties props = mailSender.getJavaMailProperties();
        props.put("mail.smtp.auth", auth);
        props.put("mail.smtp.starttls.enable", starttls);

        return mailSender;
    }
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\config\SecurityConfig.java

```
package com.molsys.example2.config;

import com.molsys.example2.security.JwtAuthenticationFilter;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.Customizer;
import
org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
```

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

@Configuration
@EnableMethodSecurity
public class SecurityConfig {

    private final JwtAuthenticationFilter jwtAuthFilter;

    public SecurityConfig(JwtAuthenticationFilter jwtAuthFilter) {
        this.jwtAuthFilter = jwtAuthFilter;
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
                .csrf(csrf -> csrf.disable())
.sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .authorizeHttpRequests(auth -> auth
.requestMatchers("/api/auth/register", "/api/auth/login", "/api/auth/refresh-token").permitAll()
.requestMatchers("/api/auth/forgot-password", "/api/auth/reset-password").permitAll()
                        .requestMatchers("/api/auth/change-password").authenticated()
                        .anyRequest().authenticated()
                )
                .addFilterBefore(jwtAuthFilter, UsernamePasswordAuthenticationFilter.class)
                .httpBasic(Customizer.withDefaults());

        return http.build();
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    @Bean
    public AuthenticationManager authManager(AuthenticationConfiguration config) throws Exception {
        return config.getAuthenticationManager();
    }
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\controller\AuthController.java

```
package com.molsys.example2.controller;

import com.molsys.example2.dto.*;
import com.molsys.example2.service.AuthService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.web.bind.annotation.*;

@RestController
@RequestMapping("/api/auth")
@RequiredArgsConstructor
public class AuthController {
    private final AuthService authService;
    @PostMapping("/register")
    public ResponseEntity<AuthResponse> register(@RequestBody AuthRequest request) {
        return ResponseEntity.ok(authService.register(request));
    }

    @PostMapping("/login")
    public ResponseEntity<AuthResponse> login(@RequestBody AuthRequest request) {
        return ResponseEntity.ok(authService.login(request));
    }

    @PostMapping("/refresh-token")
    public ResponseEntity<AuthResponse> refresh(@RequestBody String refreshToken) {
        return ResponseEntity.ok(authService.refreshToken(refreshToken));
    }

    @PostMapping("/logout")
```

```
    public ResponseEntity<String> logout(@RequestBody String refreshToken) {
        authService.logout(refreshToken);
        return ResponseEntity.ok("Logged out successfully");
    }

    @PostMapping("/change-password")
    public ResponseEntity<String> changePassword(@RequestBody PasswordChangeRequest request) {
        Authentication authentication = SecurityContextHolder.getContext().getAuthentication();
        String email = authentication.getName();
        authService.changePassword(email, request);
        return ResponseEntity.ok("Password changed successfully");
    }

    // Password Reset Flow Endpoints
    @PostMapping("/forgot-password")
    public ResponseEntity<String> forgotPassword(@RequestBody ForgotPasswordRequest request) {
        authService.initiatePasswordReset(request.getEmail());
        // Always return success even if email doesn't exist (security best practice)
return ResponseEntity.ok("If the email exists in our system, you will receive password reset
instructions");
    }

    @PostMapping("/reset-password")
    public ResponseEntity<String> resetPassword(@RequestBody ResetPasswordRequest request) {
        authService.resetPassword(request);
        return ResponseEntity.ok("Password has been reset successfully");
    }
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\controller\CommentController.java

```
package com.molsys.example2.controller;
import com.molsys.example2.Entity.Comment;
import com.molsys.example2.Entity.User;
import com.molsys.example2.Repository.UserRepository;
import com.molsys.example2.dto.CommentResponse;
import com.molsys.example2.service.CommentService;
import lombok.RequiredArgsConstructor;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.*;
import java.util.List;
import java.util.stream.Collectors;
@RestController
@RequestMapping("/api/comments")
@RequiredArgsConstructor
public class CommentController {
    private final CommentService commentService;
    private final UserRepository userRepository;
    @GetMapping("/post/{postId}")
    public List<CommentResponse> getCommentsForPost(@PathVariable Long postId) {
        return commentService.getCommentsByPostId(postId).stream()
                .map(comment -> new CommentResponse(
                        comment.getId(),
                        comment.getMessage(),
                        comment.getPost() != null ? comment.getPost().getId() : null,
                        comment.getUser() != null ? comment.getUser().getId() : null
                ))
                .collect(Collectors.toList());
    }
    @PostMapping
    public CommentResponse addComment(@RequestBody Comment comment, Authentication authentication) {
        String email = authentication.getName();
        User user = userRepository.findByEmail(email)
                .orElseThrow(() -> new RuntimeException("User not found"));
        comment.setUser(user);
        Comment saved = commentService.addComment(comment);

        return new CommentResponse(
                saved.getId(),
                saved.getMessage(),
                saved.getPost() != null ? saved.getPost().getId() : null,
                user.getId()
        );
```

```
    }
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\controller\PostController.java

```
package com.molsys.example2.controller;

import com.molsys.example2.Entity.Post;
import com.molsys.example2.Entity.User;
import com.molsys.example2.Repository.UserRepository;
import com.molsys.example2.dto.PostResponse;
import com.molsys.example2.service.PostService;
import lombok.RequiredArgsConstructor;
import org.springframework.security.core.Authentication;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.stream.Collectors;

@RestController
@RequestMapping("/api/posts")
@RequiredArgsConstructor
public class PostController {

    private final PostService postService;
    private final UserRepository userRepository;

    @GetMapping
    public List<PostResponse> getAllPosts() {
        return postService.getAllPosts().stream()
                .map(post -> new PostResponse(
                        post.getId(),
                        post.getTitle(),
                        post.getContent(),
                        post.getUser() != null ? post.getUser().getId() : null
                ))
                .collect(Collectors.toList());
    }

    @PostMapping
    public PostResponse createPost(@RequestBody Post post, Authentication authentication) {
        String email = authentication.getName();
        User user = userRepository.findByEmail(email)
                .orElseThrow(() -> new RuntimeException("User not found"));

        post.setUser(user);
        Post savedPost = postService.createPost(post);

        return new PostResponse(
                savedPost.getId(),
                savedPost.getTitle(),
                savedPost.getContent(),
                user.getId()
        );
    }

    @GetMapping("/search")
    public List<PostResponse> searchPosts(@RequestParam("keyword") String keyword) {
        return postService.searchPosts(keyword).stream()
                .map(post -> new PostResponse(
                        post.getId(),
                        post.getTitle(),
                        post.getContent(),
                        post.getUser() != null ? post.getUser().getId() : null
                ))
                .collect(Collectors.toList());
    }
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\dto\AuthRequest.java

```
package com.molsys.example2.dto;

import com.molsys.example2.Entity.Role;
```

```
import lombok.Data;

@Data
public class AuthRequest {
    private String email;
    private String password;
    private Role role;  // Added role field
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\dto\AuthResponse.java

```
package com.molsys.example2.dto;

import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor
public class AuthResponse {
    private String accessToken;
    private String refreshToken;
    private String role;
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\dto\CommentResponse.java

```
package com.molsys.example2.dto;

import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor
public class CommentResponse {
    private Long id;
    private String message;
    private Long postId;
    private Long userId;
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\dto\ForgotPasswordRequest.java

```
package com.molsys.example2.dto;

import lombok.Data;

@Data
public class ForgotPasswordRequest {
    private String email;
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\dto\PasswordChangeRequest.java

```
package com.molsys.example2.dto;

import lombok.Data;

@Data
public class PasswordChangeRequest {
    private String currentPassword;
    private String newPassword;
    private String confirmNewPassword;
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\dto\PostResponse.java

```
package com.molsys.example2.dto;

import lombok.AllArgsConstructor;
import lombok.Data;

@Data
```

```
@AllArgsConstructor
public class PostResponse {
    private Long id;
    private String title;
    private String content;
    private Long userId;
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\dto\ResetPasswordRequest.java

```
package com.molsys.example2.dto;

import lombok.Data;


@Data
public class ResetPasswordRequest {
    private String token;
    private String newPassword;
    private String confirmPassword;
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\Entity\Comment.java

```
package com.molsys.example2.Entity;

import jakarta.persistence.*;
import lombok.*;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Comment {

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

    private String message;

    @ManyToOne
    @JoinColumn(name = "post_id")
    private Post post;

    @ManyToOne
    @JoinColumn(name = "user_id")
    private User user;
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\Entity\Post.java

```
package com.molsys.example2.Entity;

import jakarta.persistence.*;
import lombok.*;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Post {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;

    @Column(length = 1000)
    private String content;

    @ManyToOne
```

```
    @JoinColumn(name = "user_id")
    private User user;  // Foreign key to User
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\Entity\Role.java

```
package com.molsys.example2.Entity;

public enum Role {
    USER,
    ADMIN
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\Entity\User.java

```
package com.molsys.example2.Entity;

import jakarta.persistence.*;
import lombok.*;
import java.time.Instant;

@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @Column(unique = true)
    private String email;

    private String password;

    @Enumerated(EnumType.STRING)
    private Role role = Role.ADMIN;

    private String refreshToken;

    private Instant refreshTokenExpiry;

    // Password reset fields
    private String passwordResetToken;

    private Instant passwordResetTokenExpiry;
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\Repository\CommentRepository.java

```
package com.molsys.example2.Repository;

import com.molsys.example2.Entity.Comment;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface CommentRepository extends JpaRepository<Comment, Long> {
    List<Comment> findByPostId(Long postId);
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\Repository\PostRepository.java

```
package com.molsys.example2.Repository;

import com.molsys.example2.Entity.Post;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;
public interface PostRepository extends JpaRepository<Post, Long> {
```

```
    // Search by title containing a keyword (case-insensitive)
    List<Post> findByTitleContainingIgnoreCase(String keyword);

}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\Repository\UserRepository.java

```
package com.molsys.example2.Repository;

import com.molsys.example2.Entity.User;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.Optional;

public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByEmail(String email);
    Optional<User> findByRefreshToken(String token);
    Optional<User> findByPasswordResetToken(String token);
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\security\JwtAuthenticationFilter.java

```
package com.molsys.example2.security;

import com.molsys.example2.Entity.User;
import com.molsys.example2.Repository.UserRepository;
import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jws;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.util.StringUtils;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;
import java.util.Collections;

@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends OncePerRequestFilter {

    private final JwtService jwtService;
    private final UserRepository userRepository;

    @Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
FilterChain filterChain)
            throws ServletException, IOException {
        try {
            String jwt = getJwtFromRequest(request);

            if (StringUtils.hasText(jwt)) {
                try {
                    Jws<Claims> claims = jwtService.validateAccessToken(jwt);
                    Claims body = claims.getBody();

                    Long userId = body.get("id", Integer.class).longValue();
                    String role = body.get("role", String.class);

                    // Find user by ID (optional, depending on your auth needs)
                    User user = userRepository.findById(userId).orElse(null);

                    if (user != null) {
                        // Set up the authentication context
UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(
                                user.getEmail(), null,
```

```
Collections.singletonList(new SimpleGrantedAuthority("ROLE_" + role))
                        );

authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
                        SecurityContextHolder.getContext().setAuthentication(authentication);
                    }
                } catch (Exception e) {
                    // Token validation failed, don't set the security context
                    logger.error("Could not set user authentication in security context", e);
                }
            }
        } catch (Exception ex) {
            logger.error("Could not set user authentication in security context", ex);
        }

        filterChain.doFilter(request, response);
    }

    private String getJwtFromRequest(HttpServletRequest request) {
        String bearerToken = request.getHeader("Authorization");
        if (StringUtils.hasText(bearerToken) && bearerToken.startsWith("Bearer ")) {
            return bearerToken.substring(7);
        }
        return null;
    }
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\security\JwtService.java

```
package com.molsys.example2.security;

import java.nio.charset.StandardCharsets;
import java.util.Base64;
import java.util.Date;

import javax.crypto.SecretKey;

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

import com.molsys.example2.Entity.Role;

import io.jsonwebtoken.Claims;
import io.jsonwebtoken.Jws;
import io.jsonwebtoken.Jwts;
import io.jsonwebtoken.security.Keys;

@Service
public class JwtService {

    @Value("${jwt.secret}")
    private String jwtSecret;

    @Value("${jwt.expiration}")
    private Long jwtExpirationMs;

    @Value("${jwt.refresh.secret}")
    private String jwtRefreshSecret;

    @Value("${jwt.refresh.expiration}")
    private Long jwtRefreshExpirationMs;

    // Generate a secure SecretKey for access tokens
    private SecretKey getAccessTokenKey() {
        // Use the recommended method to generate a secure key
        // This ensures the key is of proper size (256 bits or more)
        byte[] keyBytes = Base64.getDecoder().decode(getSecureBase64Key(jwtSecret));
        return Keys.hmacShaKeyFor(keyBytes);
    }

    // Generate a secure SecretKey for refresh tokens
    private SecretKey getRefreshTokenKey() {
        // Use the recommended method to generate a secure key
```

```
        byte[] keyBytes = Base64.getDecoder().decode(getSecureBase64Key(jwtRefreshSecret));
        return Keys.hmacShaKeyFor(keyBytes);
    }

    // Ensure the key is properly formatted and of sufficient length
    private String getSecureBase64Key(String originalKey) {
        // If the key is already Base64 and of sufficient length, use it
        // Otherwise, pad it to ensure it's at least 256 bits (32 bytes) when decoded
        try {
            byte[] decodedKey = Base64.getDecoder().decode(originalKey);
            if (decodedKey.length >= 32) {
                return originalKey;
            }
        } catch (IllegalArgumentException e) {
            // Not a valid Base64 string, will create a new one
        }

        // Create a key that's at least 256 bits
        String paddedKey = originalKey;
        while (paddedKey.getBytes(StandardCharsets.UTF_8).length < 32) {
            paddedKey += originalKey;
        }
        return Base64.getEncoder().encodeToString(paddedKey.getBytes(StandardCharsets.UTF_8));
    }

    public String generateAccessToken(Long userId, Role role) {
        return Jwts.builder()
                .claim("id", userId)
                .claim("role", role.name())
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + jwtExpirationMs))
                .signWith(getAccessTokenKey())
                .compact();
    }

    public String generateRefreshToken(Long userId) {
        return Jwts.builder()
                .claim("id", userId)
                .setIssuedAt(new Date())
                .setExpiration(new Date(System.currentTimeMillis() + jwtRefreshExpirationMs))
                .signWith(getRefreshTokenKey())
                .compact();
    }

    public Jws<Claims> validateAccessToken(String token) {
        return Jwts.parser().setSigningKey(getAccessTokenKey()).parseClaimsJws(token);
    }

    public Jws<Claims> validateRefreshToken(String token) {
        return Jwts.parser().setSigningKey(getRefreshTokenKey()).parseClaimsJws(token);
    }
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\service\AuthService.java

```
package com.molsys.example2.service;

import com.molsys.example2.Entity.Role;
import com.molsys.example2.Entity.User;
import com.molsys.example2.Repository.UserRepository;
import com.molsys.example2.dto.*;
import com.molsys.example2.security.JwtService;
import lombok.RequiredArgsConstructor;
import lombok.extern.slf4j.Slf4j;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.time.Instant;
import java.util.Objects;
import java.util.Optional;
import java.util.UUID;
```

```
@Service
@RequiredArgsConstructor
@Slf4j

public class AuthService {
    private final UserRepository userRepo;
    private final PasswordEncoder passwordEncoder;
    private final JwtService jwtService;
    private final EmailService emailService;

    // Token expiry time - 15 minutes
    private static final long PASSWORD_RESET_TOKEN_EXPIRY = 15 * 60 * 1000;

    public AuthResponse register(AuthRequest request) {
        if (userRepo.findByEmail(request.getEmail()).isPresent()) {
            throw new RuntimeException("User already exists");
        }

        // Use the role from request if provided, otherwise default to USER
        Role userRole = (request.getRole() != null) ? request.getRole() : Role.USER;

        // Save user first to get a generated ID
        User user = User.builder()
                .email(request.getEmail())
                .name("User")
                .password(passwordEncoder.encode(request.getPassword()))
                .role(userRole) // Use the role from request or default
                .build();

        user = userRepo.save(user); // Save to assign ID

        // Now generate tokens using saved user
        String accessToken = jwtService.generateAccessToken(user.getId(), user.getRole());
        String refreshToken = jwtService.generateRefreshToken(user.getId());

        user.setRefreshToken(refreshToken);
        user.setRefreshTokenExpiry(Instant.now().plusMillis(7 * 24 * 60 * 60 * 1000));
        userRepo.save(user); // Save refresh token and expiry

        return new AuthResponse(accessToken, refreshToken, user.getRole().name());
    }

    public AuthResponse login(AuthRequest request) {
        User user = userRepo.findByEmail(request.getEmail())
                .orElseThrow(() -> new RuntimeException("Invalid credentials"));

        if (!passwordEncoder.matches(request.getPassword(), user.getPassword())) {
            throw new RuntimeException("Invalid credentials");
        }

        String accessToken = jwtService.generateAccessToken(user.getId(), user.getRole());
        String refreshToken = jwtService.generateRefreshToken(user.getId());

        user.setRefreshToken(refreshToken);
        user.setRefreshTokenExpiry(Instant.now().plusMillis(7 * 24 * 60 * 60 * 1000));
        userRepo.save(user);

        return new AuthResponse(accessToken, refreshToken, user.getRole().name());
    }

    public AuthResponse refreshToken(String token) {
        var claims = jwtService.validateRefreshToken(token).getBody();
        Long userId = claims.get("id", Integer.class).longValue();

User user = userRepo.findById(userId).orElseThrow(() -> new RuntimeException("User not found"));
        if (!token.equals(user.getRefreshToken())) {
            throw new RuntimeException("Invalid refresh token");
        }

        String newAccess = jwtService.generateAccessToken(user.getId(), user.getRole());
        return new AuthResponse(newAccess, token, user.getRole().name());
    }
```

```
    public void logout(String refreshToken) {
        Optional<User> user = userRepo.findByRefreshToken(refreshToken);
        user.ifPresent(u -> {
            u.setRefreshToken(null);
            u.setRefreshTokenExpiry(null);
            userRepo.save(u);
        });
    }

    public void changePassword(String email, PasswordChangeRequest request) {
        // Validate request
        if (request.getNewPassword() == null || request.getNewPassword().trim().isEmpty()) {
            throw new IllegalArgumentException("New password cannot be empty");
        }

        if (!Objects.equals(request.getNewPassword(), request.getConfirmNewPassword())) {
            throw new IllegalArgumentException("New password and confirm password do not match");
        }

        // Find user
        User user = userRepo.findByEmail(email)
                .orElseThrow(() -> new RuntimeException("User not found"));

        // Verify current password
        if (!passwordEncoder.matches(request.getCurrentPassword(), user.getPassword())) {
            throw new BadCredentialsException("Current password is incorrect");
        }

        // Check if new password is different from current
        if (passwordEncoder.matches(request.getNewPassword(), user.getPassword())) {
throw new IllegalArgumentException("New password must be different from current password");
        }

        // Update password
        user.setPassword(passwordEncoder.encode(request.getNewPassword()));

        // Invalidate refresh tokens for security
        user.setRefreshToken(null);
        user.setRefreshTokenExpiry(null);

        // Save changes
        userRepo.save(user);
    }

    // Password Reset Methods
    @Transactional
    public void initiatePasswordReset(String email) {
        Optional<User> userOptional = userRepo.findByEmail(email);

        // If user exists, generate and send token
        userOptional.ifPresent(user -> {
            try {
                String token = generatePasswordResetToken();
                user.setPasswordResetToken(token);
                user.setPasswordResetTokenExpiry(Instant.now().plusMillis(PASSWORD_RESET_TOKEN_EXPIR
Y));
                userRepo.save(user);

                // Send email with reset link
                emailService.sendPasswordResetEmail(user.getEmail(), token);
                log.info("Password reset initiated for user: {}", email);
            } catch (Exception e) {
                log.error("Error initiating password reset for user: {}", email, e);
                throw new RuntimeException("Failed to initiate password reset", e);
            }
        });
    }

    @Transactional
    public void resetPassword(ResetPasswordRequest request) {
        if (request.getToken() == null || request.getToken().trim().isEmpty()) {
            throw new IllegalArgumentException("Reset token cannot be empty");
```

```
        }

        if (request.getNewPassword() == null || request.getNewPassword().trim().isEmpty()) {
            throw new IllegalArgumentException("New password cannot be empty");
        }

        if (!Objects.equals(request.getNewPassword(), request.getConfirmPassword())) {
            throw new IllegalArgumentException("Passwords do not match");
        }

        User user = userRepo.findByPasswordResetToken(request.getToken())
                .orElseThrow(() -> new RuntimeException("Invalid or expired token"));

        // Check token expiry
        if (user.getPasswordResetTokenExpiry() == null ||
                user.getPasswordResetTokenExpiry().isBefore(Instant.now())) {
            throw new RuntimeException("Password reset token has expired");
        }

        // Update password
        user.setPassword(passwordEncoder.encode(request.getNewPassword()));

        // Clear reset token and invalidate any existing sessions for security
        user.setPasswordResetToken(null);
        user.setPasswordResetTokenExpiry(null);
        user.setRefreshToken(null);
        user.setRefreshTokenExpiry(null);

        userRepo.save(user);
        log.info("Password reset completed for user: {}", user.getEmail());
    }

    private String generatePasswordResetToken() {
        return UUID.randomUUID().toString();
    }
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\service\CommentService.java

```
package com.molsys.example2.service;

import com.molsys.example2.Entity.Comment;
import com.molsys.example2.Repository.CommentRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class CommentService {
    private final CommentRepository commentRepository;

    public List<Comment> getCommentsByPostId(Long postId) {
        return commentRepository.findByPostId(postId);
    }

    public Comment addComment(Comment comment) {
        return commentRepository.save(comment);
    }
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\service\EmailService.java

```
package com.molsys.example2.service;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSender;
import org.springframework.stereotype.Service;
import lombok.extern.slf4j.Slf4j;

@Service
```

```
@Slf4j
public class EmailService {
    private final JavaMailSender mailSender;

    @Value("${spring.mail.username}")
    private String fromEmail;

    @Value("${app.frontend-url}")
    private String frontendUrl;

    public EmailService(JavaMailSender mailSender) {
        this.mailSender = mailSender;
    }

    public void sendPasswordResetEmail(String to, String token) {
        try {
            SimpleMailMessage message = new SimpleMailMessage();
            message.setFrom(fromEmail);
            message.setTo(to);
            message.setSubject("Password Reset Request");
            String resetUrl = frontendUrl + "/reset-password?token=" + token;
            message.setText("Hello,\n\n" +
"You have requested to reset your password. Please click on the link below to reset your
password:\n\n" +
                    resetUrl + "\n\n" +
                    "If you did not request a password reset, please ignore this email.\n\n" +
                    "This link will expire in 15 minutes.\n\n" +
                    "Best regards,\nYour Application Team");

            mailSender.send(message);
            log.info("Password reset email sent to: {}", to);
        } catch (Exception e) {
            log.error("Failed to send password reset email to: {}", to, e);
            throw new RuntimeException("Failed to send password reset email", e);
        }
    }
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\main\java\com molsys\example2\service\PostService.java

```
package com.molsys.example2.service;

import com.molsys.example2.Entity.Post;
import com.molsys.example2.Repository.PostRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class PostService {
    private final PostRepository postRepository;

    public List<Post> getAllPosts() {
        return postRepository.findAll();
    }

    public Post createPost(Post post) {
        return postRepository.save(post);
    }
    public List<Post> searchPosts(String keyword) {
        return postRepository.findByTitleContainingIgnoreCase(keyword);
    }

}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\test\java\com molsys\example2\AuthServicePasswordResetTest.java

```
package com.molsys.example2.service;

import com.molsys.example2.Entity.User;
import com.molsys.example2.Repository.UserRepository;
```

```
import com.molsys.example2.dto.ResetPasswordRequest;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import org.springframework.security.crypto.password.PasswordEncoder;

import java.time.Instant;
import java.util.Optional;

import static org.junit.jupiter.api.Assertions.*;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.Mockito.*;

class AuthServicePasswordResetTest {

    @Mock
    private UserRepository userRepository;

    @Mock
    private PasswordEncoder passwordEncoder;

    @Mock
    private EmailService emailService;

    @InjectMocks
    private AuthService authService;

    private User testUser;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);

        testUser = new User();
        testUser.setId(1L);
        testUser.setEmail("test@example.com");
        testUser.setPassword("encoded_password");
    }

    @Test
    void initiatePasswordReset_ShouldGenerateTokenAndSendEmail() {
        // Arrange
        when(userRepository.findByEmail("test@example.com")).thenReturn(Optional.of(testUser));

        // Act
        authService.initiatePasswordReset("test@example.com");

        // Assert
        verify(userRepository).save(any(User.class));
        verify(emailService).sendPasswordResetEmail(eq("test@example.com"), any(String.class));

        // Capture the saved user to verify token was set
        verify(userRepository).save(argThat(user ->
                user.getPasswordResetToken() != null &&
                        user.getPasswordResetTokenExpiry() != null
        ));
    }

    @Test
    void resetPassword_WithValidTokenAndMatchingPasswords_ShouldUpdatePassword() {
        // Arrange
        String validToken = "valid_token";
        testUser.setPasswordResetToken(validToken);
        testUser.setPasswordResetTokenExpiry(Instant.now().plusSeconds(900)); // 15 minutes from now

        when(userRepository.findByPasswordResetToken(validToken)).thenReturn(Optional.of(testUser));
        when(passwordEncoder.encode("newPassword")).thenReturn("new_encoded_password");

        ResetPasswordRequest request = new ResetPasswordRequest();
        request.setToken(validToken);
        request.setNewPassword("newPassword");
```

```
        request.setConfirmPassword("newPassword");

        // Act
        authService.resetPassword(request);

        // Assert
        verify(userRepository).save(argThat(user ->
                user.getPassword().equals("new_encoded_password") &&
                        user.getPasswordResetToken() == null &&
                        user.getPasswordResetTokenExpiry() == null &&
                        user.getRefreshToken() == null &&
                        user.getRefreshTokenExpiry() == null
        ));
    }

    @Test
    void resetPassword_WithExpiredToken_ShouldThrowException() {
        // Arrange
        String expiredToken = "expired_token";
        testUser.setPasswordResetToken(expiredToken);
        testUser.setPasswordResetTokenExpiry(Instant.now().minusSeconds(60)); // 1 minute ago

        when(userRepository.findByPasswordResetToken(expiredToken)).thenReturn(Optional.of(testUser)
);

        ResetPasswordRequest request = new ResetPasswordRequest();
        request.setToken(expiredToken);
        request.setNewPassword("newPassword");
        request.setConfirmPassword("newPassword");

        // Act & Assert
        Exception exception = assertThrows(RuntimeException.class, () -> {
            authService.resetPassword(request);
        });

        assertEquals("Password reset token has expired", exception.getMessage());
        verify(userRepository, never()).save(any(User.class));
    }

    @Test
    void resetPassword_WithNonMatchingPasswords_ShouldThrowException() {
        // Arrange
        String validToken = "valid_token";
        testUser.setPasswordResetToken(validToken);
        testUser.setPasswordResetTokenExpiry(Instant.now().plusSeconds(900));

        when(userRepository.findByPasswordResetToken(validToken)).thenReturn(Optional.of(testUser));

        ResetPasswordRequest request = new ResetPasswordRequest();
        request.setToken(validToken);
        request.setNewPassword("newPassword");
        request.setConfirmPassword("differentPassword");

        // Act & Assert
        Exception exception = assertThrows(IllegalArgumentException.class, () -> {
            authService.resetPassword(request);
        });

        assertEquals("Passwords do not match", exception.getMessage());
        verify(userRepository, never()).save(any(User.class));
    }
}
```

File: C:\Users\hp\OneDrive\Desktop\Molsys Internship\day 1 spring boot\example2\src\test\java\com molsys\example2\Example2ApplicationTests.java

```
package com.molsys.example2;

import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
class Example2ApplicationTests {

■@Test
```

```
■void contextLoads() {
■}

}
```