■ Lost & Found Backend - Java Code Export

■ Backend Java Files:

[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\LostAndFoundApplication.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\annotation\RateLimit.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\aspect\RateLimitAspect.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\config\FileStorageProperties.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\config\ModelMapperConfig.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\config\RateLimitConfig.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\config\SecurityConfig.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\config\WebConfig.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\controller\AdminController.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\controller\AuthController.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\controller\ClaimController.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\controller\DashboardController.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\controller\FeedbackController.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\controller\ItemController.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\controller\MessageController.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\controller\RateLimitController.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\dto\request\FeedbackRequest.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\dto\request\ItemRequest.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\dto\request\LoginRequest.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\dto\request\MessageRequest.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\dto\request\RegisterRequest.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\dto\request\TokenRefreshRequest.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\dto\response\ApiResponse.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\dto\response\AuthResponse.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\dto\response\ClaimResponse.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\dto\response\DashboardResponse.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\dto\response\FeedbackResponse.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\dto\response\ItemResponse.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\dto\response\MessageResponse.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\dto\response\TokenRefreshResponse.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\exception\BadRequestException.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\exception\GlobalExceptionHandler.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\exception\ResourceNotFoundException.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\exception\UnauthorizedException.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\model\Claim.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\model\Feedback.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\model\Item.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\model\Message.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\model\RefreshToken.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\model\User.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\repository\ClaimRepository.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\repository\FeedbackRepository.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\repository\ItemRepository.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\repository\MessageRepository.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\repository\RefreshTokenRepository.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\repository\UserRepository.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\scheduler\ClaimCleanupScheduler.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\scheduler\TokenCleanupScheduler.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\security\CustomUserDetailsService.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\security\JwtAuthenticationFilter.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\security\JwtTokenProvider.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\security\RateLimitFilter.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\security\UserPrincipal.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\service\ClaimService.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\service\FeedbackService.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\service\FileStorageService.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\service\ItemService.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\service\MessageService.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\service\RefreshTokenService.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\service\UserService.java
[JAVA] lost-and-found-backend\src\main\java\com\lostandfound\util\CookieUtil.java

■ File: lost-and-found-backend\src\main\java\com\lostandfound\LostAndFoundApplication.java

```
  1: package com.lostandfound;
  2:
  3: import org.springframework.boot.SpringApplication;
  4: import org.springframework.boot.autoconfigure.SpringBootApplication;
  5: import org.springframework.scheduling.annotation.EnableScheduling;
  6:
  7: @SpringBootApplication
  8: @EnableScheduling
  9: public class LostAndFoundApplication {
 10:     public static void main(String[] args) {
 11:         SpringApplication.run(LostAndFoundApplication.class, args);
 12:     }
 13: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\annotation\RateLimit.java

```
  1: package com.lostandfound.annotation;
  2:
  3: import com.lostandfound.config.RateLimitConfig;
  4:
  5: import java.lang.annotation.ElementType;
  6: import java.lang.annotation.Retention;
  7: import java.lang.annotation.RetentionPolicy;
  8: import java.lang.annotation.Target;
  9:
 10: /**
 11:  * Custom annotation for method-level rate limiting
 12:  * Usage: @RateLimit(type = RateLimitConfig.RateLimitType.AUTH)
 13:  */
 14: @Target({ElementType.METHOD, ElementType.TYPE})
 15: @Retention(RetentionPolicy.RUNTIME)
 16: public @interface RateLimit {
 17:     RateLimitConfig.RateLimitType type() default RateLimitConfig.RateLimitType.API;
 18: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\aspect\RateLimitAspect.java

```
  1: package com.lostandfound.aspect;
  2:
  3: import com.lostandfound.annotation.RateLimit;
  4: import com.lostandfound.config.RateLimitConfig;
  5: import com.lostandfound.exception.BadRequestException;
  6: import io.github.bucket4j.Bucket;
  7: import io.github.bucket4j.ConsumptionProbe;
  8: import jakarta.servlet.http.HttpServletRequest;
  9: import lombok.RequiredArgsConstructor;
 10: import org.aspectj.lang.ProceedingJoinPoint;
 11: import org.aspectj.lang.annotation.Around;
 12: import org.aspectj.lang.annotation.Aspect;
 13: import org.slf4j.Logger;
 14: import org.slf4j.LoggerFactory;
 15: import org.springframework.stereotype.Component;
 16: import org.springframework.web.context.request.RequestContextHolder;
 17: import org.springframework.web.context.request.ServletRequestAttributes;
 18:
 19: /**
 20:  * AOP Aspect for method-level rate limiting using @RateLimit annotation
 21:  * This is optional and works alongside the filter-based rate limiting
 22:  */
 23: @Aspect
 24: @Component
 25: @RequiredArgsConstructor
 26: public class RateLimitAspect {
 27:
 28:     private static final Logger logger = LoggerFactory.getLogger(RateLimitAspect.class);
 29:
 30:     private final RateLimitConfig rateLimitConfig;
 31:
 32:     @Around("@annotation(rateLimit)")
 33:     public Object rateLimit(ProceedingJoinPoint joinPoint, RateLimit rateLimit) throws Thr
    | owable {
 34:         HttpServletRequest request = getCurrentRequest();
 35:
 36:         if (request == null) {
 37:             // If no request context, skip rate limiting
 38:             return joinPoint.proceed();
 39:         }
 40:
 41:         String clientIp = getClientIP(request);
```

```
 42:         String method = joinPoint.getSignature().getName();
 43:
 44:         RateLimitConfig.RateLimitType limitType = rateLimit.type();
 45:         String bucketKey = clientIp + ":" + method + ":" + limitType.name();
 46:
 47:         Bucket bucket = rateLimitConfig.resolveBucket(bucketKey, limitType);
 48:         ConsumptionProbe probe = bucket.tryConsumeAndReturnRemaining(1);
 49:
 50:         if (probe.isConsumed()) {
 51:             logger.debug("Rate limit check passed for IP: {} on method: {}", clientIp, met
    | hod);
 52:             return joinPoint.proceed();
 53:         } else {
 54:             long waitForRefill = probe.getNanosToWaitForRefill() / 1_000_000_000;
 55:             logger.warn("Rate limit exceeded for IP: {} on method: {}", clientIp, method);
 56:             throw new BadRequestException(
 57:                     "Rate limit exceeded. Please try again in " + waitForRefill + " second
    | s."
 58:             );
 59:         }
 60:     }
 61:
 62:     private HttpServletRequest getCurrentRequest() {
 63:         ServletRequestAttributes attributes =
 64:                 (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();
 65:         return attributes != null ? attributes.getRequest() : null;
 66:     }
 67:
 68:     private String getClientIP(HttpServletRequest request) {
 69:         String xfHeader = request.getHeader("X-Forwarded-For");
 70:         String remoteAddr = request.getRemoteAddr();
 71:         boolean isTrustedProxy = isTrustedProxy(remoteAddr);
 72:
 73:         if (isTrustedProxy && xfHeader != null && !xfHeader.isEmpty()) {
 74:             String clientIp = xfHeader.split(",")[0].trim();
 75:             if (isValidIP(clientIp)) {
 76:                 return clientIp;
 77:             }
 78:         }
 79:
 80:         return remoteAddr;
 81:     }
 82:
 83:     private boolean isTrustedProxy(String ip) {
 84:         return "127.0.0.1".equals(ip) ||
 85:                 "0:0:0:0:0:0:0:1".equals(ip) ||
 86:                 "::1".equals(ip);
 87:     }
 88:
 89:     private boolean isValidIP(String ip) {
 90:         if (ip == null || ip.isEmpty()) {
 91:             return false;
 92:         }
 93:         String ipv4Pattern = "^((25[0-5]|(2[0-4]|1\\d|[1-9]|)\\d)\\.?\\b){4}$";
 94:         String ipv6Pattern = "^(([0-9a-fA-F]{1,4}:){7}[0-9a-fA-F]{1,4}|([0-9a-fA-F]{1,4}:)
    | {1,7}:|([0-9a-fA-F]{1,4}:){1,6}:[0-9a-fA-F]{1,4}|([0-9a-fA-F]{1,4}:){1,5}(:[0-9a-fA-F]{1,4
    | }){1,2})$";
 95:         return ip.matches(ipv4Pattern) || ip.matches(ipv6Pattern);
 96:     }
 97: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\config\FileStorageProperties.java

```
  1: package com.lostandfound.config;
  2:
  3: import org.springframework.boot.context.properties.ConfigurationProperties;
  4: import org.springframework.stereotype.Component;
  5:
  6: import lombok.Data;
  7:
  8: @Component
  9: @ConfigurationProperties(prefix = "file")
 10: @Data
 11: public class FileStorageProperties {
 12:     private String uploadDir;
 13: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\config\ModelMapperConfig.java

```
  1: package com.lostandfound.config;
  2:
```

```
 3: import org.modelmapper.ModelMapper;
 4: import org.modelmapper.convention.MatchingStrategies;
 5: import org.springframework.context.annotation.Bean;
 6: import org.springframework.context.annotation.Configuration;
 7:
 8: @Configuration
 9: public class ModelMapperConfig {
10:
11:     @Bean
12:     public ModelMapper modelMapper() {
13:         ModelMapper modelMapper = new ModelMapper();
14:         modelMapper.getConfiguration()
15:                 .setMatchingStrategy(MatchingStrategies.STRICT)
16:                 .setSkipNullEnabled(true);
17:         return modelMapper;
18:     }
19: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\config\RateLimitConfig.java

```
 1: package com.lostandfound.config;
 2:
 3: import io.github.bucket4j.Bandwidth;
 4: import io.github.bucket4j.Bucket;
 5: import io.github.bucket4j.Refill;
 6: import org.springframework.context.annotation.Bean;
 7: import org.springframework.context.annotation.Configuration;
 8:
 9: import java.time.Duration;
10: import java.util.Map;
11: import java.util.concurrent.ConcurrentHashMap;
12:
13: @Configuration
14: public class RateLimitConfig {
15:
16:     /**
17:      * In-memory cache for storing buckets per IP address
18:      * For production, consider using Redis or Hazelcast for distributed caching
19:      */
20:     private final Map<String, Bucket> cache = new ConcurrentHashMap<>();
21:
22:     /**
23:      * Rate limit configurations for different endpoints
24:      */
25:     public enum RateLimitType {
26:         // Auth endpoints - 5 requests per minute
27:         AUTH(5, Duration.ofMinutes(1)),
28:
29:         // General API endpoints - 100 requests per minute
30:         API(100, Duration.ofMinutes(1)),
31:
32:         // Admin endpoints - 50 requests per minute
33:         ADMIN(50, Duration.ofMinutes(1)),
34:
35:         // File upload - 10 requests per 5 minutes
36:         UPLOAD(10, Duration.ofMinutes(5)),
37:
38:         // Public endpoints - 200 requests per minute
39:         PUBLIC(200, Duration.ofMinutes(1));
40:
41:         private final long capacity;
42:         private final Duration refillDuration;
43:
44:         RateLimitType(long capacity, Duration refillDuration) {
45:             this.capacity = capacity;
46:             this.refillDuration = refillDuration;
47:         }
48:
49:         public long getCapacity() {
50:             return capacity;
51:         }
52:
53:         public Duration getRefillDuration() {
54:             return refillDuration;
55:         }
56:     }
57:
58:     /**
59:      * Resolve bucket for given key and rate limit type
60:      */
61:     public Bucket resolveBucket(String key, RateLimitType limitType) {
62:         return cache.computeIfAbsent(key, k -> createNewBucket(limitType));
63:     }
```

```
64:
65:     /**
66:      * Create new bucket with specified rate limit
67:      */
68:     private Bucket createNewBucket(RateLimitType limitType) {
69:         Bandwidth limit = Bandwidth.classic(
70:                 limitType.getCapacity(),
71:                 Refill.intervally(limitType.getCapacity(), limitType.getRefillDuration())
72:         );
73:         return Bucket.builder()
74:                 .addLimit(limit)
75:                 .build();
76:     }
77:
78:     /**
79:      * Clear bucket for specific key (useful for testing or admin actions)
80:      */
81:     public void clearBucket(String key) {
82:         cache.remove(key);
83:     }
84:
85:     /**
86:      * Clear all buckets (useful for testing)
87:      */
88:     public void clearAllBuckets() {
89:         cache.clear();
90:     }
91:
92:     /**
93:      * Get current cache size
94:      */
95:     public int getCacheSize() {
96:         return cache.size();
97:     }
98: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\config\SecurityConfig.java

```
  1: package com.lostandfound.config;
  2:
  3: import com.lostandfound.security.JwtAuthenticationFilter;
  4: import com.lostandfound.security.RateLimitFilter;
  5: import lombok.RequiredArgsConstructor;
  6: import org.springframework.beans.factory.annotation.Value;
  7: import org.springframework.context.annotation.Bean;
  8: import org.springframework.context.annotation.Configuration;
  9: import org.springframework.security.authentication.AuthenticationManager;
 10: import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
 11: import org.springframework.security.config.annotation.authentication.configuration.Authent
    | icationConfiguration;
 12: import org.springframework.security.config.annotation.method.configuration.EnableMethodSec
    | urity;
 13: import org.springframework.security.config.annotation.web.builders.HttpSecurity;
 14: import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
 15: import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigur
    | er;
 16: import org.springframework.security.config.http.SessionCreationPolicy;
 17: import org.springframework.security.core.userdetails.UserDetailsService;
 18: import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
 19: import org.springframework.security.crypto.password.PasswordEncoder;
 20: import org.springframework.security.web.SecurityFilterChain;
 21: import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilte
    | r;
 22: import org.springframework.security.web.csrf.CookieCsrfTokenRepository;
 23: import org.springframework.security.web.csrf.CsrfTokenRequestAttributeHandler;
 24: import org.springframework.security.web.header.writers.ReferrerPolicyHeaderWriter;
 25: import org.springframework.security.web.header.writers.StaticHeadersWriter;
 26: import org.springframework.security.web.header.writers.XXssProtectionHeaderWriter;
 27: import org.springframework.web.cors.CorsConfiguration;
 28: import org.springframework.web.cors.CorsConfigurationSource;
 29: import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
 30:
 31: import java.util.Arrays;
 32: import java.util.List;
 33:
 34: @Configuration
 35: @EnableWebSecurity
 36: @EnableMethodSecurity(prePostEnabled = true, securedEnabled = true)
 37: @RequiredArgsConstructor
 38: public class SecurityConfig {
 39:
 40:     private final JwtAuthenticationFilter jwtAuthenticationFilter;
 41:     private final RateLimitFilter rateLimitFilter;
```

```
    private final UserDetailsService userDetailsService;

    @Value("${cors.allowed-origins}")
    private String allowedOrigins;

    @Value("${security.csrf.enabled:true}")
    private boolean csrfEnabled;

    @Bean
    public PasswordEncoder passwordEncoder() {
        // Use strength 12 for production (default is 10)
        return new BCryptPasswordEncoder(12);
    }

    @Bean
    public DaoAuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService);
        authProvider.setPasswordEncoder(passwordEncoder());
        authProvider.setHideUserNotFoundExceptions(true); // Security best practice
        return authProvider;
    }

    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration authCon
   | fig)
            throws Exception {
        return authConfig.getAuthenticationManager();
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        // CSRF Token Handler for SPA
        CsrfTokenRequestAttributeHandler requestHandler = new CsrfTokenRequestAttributeHan
   | dler();
        requestHandler.setCsrfRequestAttributeName("_csrf");

        http
                // CSRF Configuration
                .csrf(csrf -> {
                    if (csrfEnabled) {
                        // Enable CSRF with Cookie-based tokens for SPA
                        csrf.csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFal
   | se())
                                .csrfTokenRequestHandler(requestHandler)
                                .ignoringRequestMatchers(
                                        "/api/auth/login",
                                        "/api/auth/register",
                                        "/api/auth/refresh"
                                ); // Exclude specific auth endpoints
                    } else {
                        // Disable CSRF for stateless JWT authentication
                        csrf.disable();
                    }
                })

                // CORS Configuration
                .cors(cors -> cors.configurationSource(corsConfigurationSource()))

                // Session Management - Stateless
                .sessionManagement(session ->
                        session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))

                // Authorization Rules
                .authorizeHttpRequests(auth -> auth
                        // Public endpoints
                        .requestMatchers(
                                "/api/auth/login",
                                "/api/auth/register",
                                "/api/auth/refresh",
                                "/uploads/**",
                                "/static/**",
                                "/error",
                                "/actuator/health",
                                "/actuator/info"
                        ).permitAll()

                        // Admin endpoints
                        .requestMatchers("/admin/**").hasRole("ADMIN")

                        // All other requests require authentication
                        .anyRequest().authenticated()
                )

                // Authentication Provider
```

```
124:                 .authenticationProvider(authenticationProvider())
125:
126:                 // Add filters in correct order:
127:                 // 1. First add JWT filter before UsernamePasswordAuthenticationFilter
128:                 .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFi
    | lter.class)
129:                 // 2. Then add Rate Limit filter before JWT filter
130:                 .addFilterBefore(rateLimitFilter, JwtAuthenticationFilter.class)
131:
132:                 // Security Headers
133:                 .headers(headers -> headers
134:                         // Content Security Policy
135:                         .contentSecurityPolicy(csp ->
136:                                 csp.policyDirectives("default-src 'self'; " +
137:                                         "script-src 'self' 'unsafe-inline'; " +
138:                                         "style-src 'self' 'unsafe-inline'; " +
139:                                         "img-src 'self' data: https:; " +
140:                                         "font-src 'self' data:; " +
141:                                         "connect-src 'self'"))
142:
143:                         // Frame Options - Prevent Clickjacking
144:                         .frameOptions(frame -> frame.deny())
145:
146:                         // XSS Protection
147:                         .xssProtection(xss -> xss
148:                                 .headerValue(XXssProtectionHeaderWriter.HeaderValue.ENABLE
    | D_MODE_BLOCK))
149:
150:                         // HSTS - Force HTTPS
151:                         .httpStrictTransportSecurity(hsts -> hsts
152:                                 .includeSubDomains(true)
153:                                 .maxAgeInSeconds(31536000)) // 1 year
154:
155:                         // Prevent MIME Sniffing
156:                         .contentTypeOptions(contentType -> {})
157:
158:                         // Referrer Policy
159:                         .referrerPolicy(referrer ->
160:                                 referrer.policy(ReferrerPolicyHeaderWriter.ReferrerPolicy.
    | STRICT_ORIGIN_WHEN_CROSS_ORIGIN))
161:
162:                         // Permissions Policy
163:                         .addHeaderWriter(new StaticHeadersWriter(
164:                                 "Permissions-Policy",
165:                                 "geolocation=(), microphone=(), camera=()"
166:                         ))
167:                 );
168:
169:         return http.build();
170:     }
171:
172:     @Bean
173:     public CorsConfigurationSource corsConfigurationSource() {
174:         CorsConfiguration configuration = new CorsConfiguration();
175:
176:         // Parse allowed origins from application.properties
177:         List<String> origins = Arrays.asList(allowedOrigins.split(","));
178:         configuration.setAllowedOriginPatterns(origins);
179:
180:         // Allowed HTTP methods
181:         configuration.setAllowedMethods(Arrays.asList(
182:                 "GET", "POST", "PUT", "DELETE", "PATCH", "OPTIONS"
183:         ));
184:
185:         // Allowed headers
186:         configuration.setAllowedHeaders(Arrays.asList(
187:                 "Authorization",
188:                 "Content-Type",
189:                 "X-Requested-With",
190:                 "Accept",
191:                 "Origin",
192:                 "Access-Control-Request-Method",
193:                 "Access-Control-Request-Headers",
194:                 "X-CSRF-TOKEN"
195:         ));
196:
197:         // Exposed headers (so frontend can read them)
198:         configuration.setExposedHeaders(Arrays.asList(
199:                 "Authorization",
200:                 "X-CSRF-TOKEN",
201:                 "Access-Control-Allow-Origin",
202:                 "Access-Control-Allow-Credentials"
203:         ));
204:
205:         // Allow credentials (cookies, authorization headers)
```

```
206:         configuration.setAllowCredentials(true);
207:
208:         // Cache preflight response for 1 hour
209:         configuration.setMaxAge(3600L);
210:
211:         UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
212:         source.registerCorsConfiguration("/**", configuration);
213:
214:         return source;
215:     }
216: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\config\WebConfig.java

```
  1: package com.lostandfound.config;
  2:
  3: import org.springframework.beans.factory.annotation.Value;
  4: import org.springframework.context.annotation.Configuration;
  5: import org.springframework.web.servlet.config.annotation.CorsRegistry;
  6: import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
  7: import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;
  8:
  9: @Configuration
 10: public class WebConfig implements WebMvcConfigurer {
 11:
 12:     @Value("${cors.allowed-origins}")
 13:     private String allowedOrigins;
 14:
 15:     @Value("${file.upload-dir}")
 16:     private String uploadDir;
 17:
 18:     @Override
 19:     public void addCorsMappings(CorsRegistry registry) {
 20:         registry.addMapping("/**")
 21:                 .allowedOrigins(allowedOrigins.split(","))
 22:                 .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
 23:                 .allowedHeaders("*")
 24:                 .allowCredentials(true)
 25:                 .maxAge(3600);
 26:     }
 27:
 28:     @Override
 29:     public void addResourceHandlers(ResourceHandlerRegistry registry) {
 30:         registry.addResourceHandler("/uploads/**")
 31:                 .addResourceLocations("file:" + uploadDir + "/");
 32:         registry.addResourceHandler("/static/**")
 33:                 .addResourceLocations("file:" + uploadDir + "/");
 34:     }
 35: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\controller\AdminController.java

```
  1: package com.lostandfound.controller;
  2:
  3: import com.lostandfound.dto.response.ApiResponse;
  4: import com.lostandfound.dto.response.ClaimResponse;
  5: import com.lostandfound.dto.response.FeedbackResponse;
  6: import com.lostandfound.dto.response.ItemResponse;
  7: import com.lostandfound.exception.BadRequestException;
  8: import com.lostandfound.exception.ResourceNotFoundException;
  9: import com.lostandfound.model.User;
 10: import com.lostandfound.repository.UserRepository;
 11: import com.lostandfound.security.UserPrincipal;
 12: import com.lostandfound.service.ClaimService;
 13: import com.lostandfound.service.FeedbackService;
 14: import com.lostandfound.service.ItemService;
 15: import com.lostandfound.service.UserService;
 16: import lombok.RequiredArgsConstructor;
 17: import org.slf4j.Logger;
 18: import org.slf4j.LoggerFactory;
 19: import org.springframework.http.ResponseEntity;
 20: import org.springframework.security.access.prepost.PreAuthorize;
 21: import org.springframework.security.core.annotation.AuthenticationPrincipal;
 22: import org.springframework.web.bind.annotation.*;
 23:
 24: import java.util.HashMap;
 25: import java.util.List;
 26: import java.util.Map;
 27: import java.util.stream.Collectors;
 28:
 29: @RestController
```

```
@RequestMapping("/admin")
@PreAuthorize("hasRole('ADMIN')")
@RequiredArgsConstructor
public class AdminController {

    private static final Logger logger = LoggerFactory.getLogger(AdminController.class);

    private final ItemService itemService;
    private final ClaimService claimService;
    private final FeedbackService feedbackService;
    private final UserRepository userRepository;
    private final UserService userService;

    @GetMapping("/dashboard")
    public ResponseEntity<Map<String, Object>> getAdminDashboard(
            @AuthenticationPrincipal UserPrincipal currentUser,
            @RequestParam(defaultValue = "0") int page,
            @RequestParam(defaultValue = "50") int size) {

        if (currentUser == null) {
            throw new BadRequestException("You must be logged in");
        }

        // Limit page size to prevent abuse
        if (size > 100) {
            size = 100;
        }

        // Get all data
        List<ItemResponse> items = itemService.searchItems("", "");
        List<ClaimResponse> claims = claimService.getUserClaims(null);
        List<FeedbackResponse> feedback = feedbackService.getAllFeedback();
        List<User> users = userRepository.findByRoleNot(User.Role.ADMIN);

        // Apply manual pagination
        int itemsStart = Math.min(page * size, items.size());
        int itemsEnd = Math.min(itemsStart + size, items.size());
        List<ItemResponse> paginatedItems = items.subList(itemsStart, itemsEnd);

        int claimsStart = Math.min(page * size, claims.size());
        int claimsEnd = Math.min(claimsStart + size, claims.size());
        List<ClaimResponse> paginatedClaims = claims.subList(claimsStart, claimsEnd);

        List<Map<String, Object>> userList = users.stream()
                .skip((long) page * size)
                .limit(size)
                .map(user -> {
                    Map<String, Object> userMap = new HashMap<>();
                    userMap.put("id", user.getId());
                    userMap.put("name", user.getName());
                    userMap.put("email", user.getEmail());
                    userMap.put("role", user.getRole().name());
                    userMap.put("createdAt", user.getCreatedAt());
                    return userMap;
                })
                .collect(Collectors.toList());

        Map<String, Object> response = new HashMap<>();
        response.put("success", true);
        response.put("items", paginatedItems);
        response.put("claims", paginatedClaims);
        response.put("users", userList);
        response.put("feedback", feedback.stream().limit(size).collect(Collectors.toList()
   | ));
        response.put("stats", Map.of(
                "totalItems", items.size(),
                "totalClaims", claims.size(),
                "totalUsers", users.size(),
                "totalFeedback", feedback.size()
        ));
        response.put("pagination", Map.of(
                "page", page,
                "size", size,
                "hasMore", itemsEnd < items.size()
        ));

        return ResponseEntity.ok(response);
    }

    @DeleteMapping("/items/{itemId}")
    public ResponseEntity<ApiResponse> deleteItem(
            @PathVariable Long itemId,
            @AuthenticationPrincipal UserPrincipal currentUser) {

        if (currentUser == null) {
```

```
            throw new BadRequestException("You must be logged in");
        }

        logger.info("Admin ID {} deleting item {}", currentUser.getId(), itemId);

        // Admin can delete any item
        itemService.deleteItemAsAdmin(itemId, currentUser);

        ApiResponse response = ApiResponse.builder()
                .success(true)
                .message("Item deleted successfully")
                .build();

        return ResponseEntity.ok(response);
    }

    @DeleteMapping("/claims/{claimId}")
    public ResponseEntity<ApiResponse> deleteClaim(
            @PathVariable Long claimId,
            @AuthenticationPrincipal UserPrincipal currentUser) {

        if (currentUser == null) {
            throw new BadRequestException("You must be logged in");
        }

        logger.info("Admin ID {} deleting claim {}", currentUser.getId(), claimId);

        claimService.deleteClaim(claimId);

        ApiResponse response = ApiResponse.builder()
                .success(true)
                .message("Claim deleted successfully")
                .build();

        return ResponseEntity.ok(response);
    }

    @DeleteMapping("/users/{userId}")
    public ResponseEntity<ApiResponse> deleteUser(
            @PathVariable Long userId,
            @AuthenticationPrincipal UserPrincipal currentUser) {

        if (currentUser == null) {
            throw new BadRequestException("You must be logged in");
        }

        logger.info("Admin ID {} attempting to delete user ID {}", currentUser.getId(), us
    | erId);

        // Use the service method to delete user with all related data
        userService.deleteUser(userId, currentUser);

        ApiResponse response = ApiResponse.builder()
                .success(true)
                .message("User deleted successfully")
                .build();

        return ResponseEntity.ok(response);
    }

    @DeleteMapping("/feedback/{feedbackId}")
    public ResponseEntity<ApiResponse> deleteFeedback(
            @PathVariable Long feedbackId,
            @AuthenticationPrincipal UserPrincipal currentUser) {

        if (currentUser == null) {
            throw new BadRequestException("You must be logged in");
        }

        logger.info("Admin ID {} deleting feedback {}", currentUser.getId(), feedbackId);

        feedbackService.deleteFeedback(feedbackId);

        ApiResponse response = ApiResponse.builder()
                .success(true)
                .message("Feedback deleted successfully")
                .build();

        return ResponseEntity.ok(response);
    }

    @GetMapping("/items")
    public ResponseEntity<Map<String, Object>> getAllItems(
            @RequestParam(required = false, defaultValue = "") String search,
            @RequestParam(required = false, defaultValue = "") String status) {
```

```
198:
199:         List<ItemResponse> items = itemService.searchItems(search, status);
200:
201:         Map<String, Object> response = new HashMap<>();
202:         response.put("success", true);
203:         response.put("message", "Items retrieved successfully");
204:         response.put("items", items);
205:         response.put("count", items.size());
206:
207:         return ResponseEntity.ok(response);
208:     }
209:
210:     @GetMapping("/claims")
211:     public ResponseEntity<Map<String, Object>> getAllClaims() {
212:         List<ClaimResponse> claims = claimService.getUserClaims(null);
213:
214:         Map<String, Object> response = new HashMap<>();
215:         response.put("success", true);
216:         response.put("message", "Claims retrieved successfully");
217:         response.put("claims", claims);
218:         response.put("count", claims.size());
219:
220:         return ResponseEntity.ok(response);
221:     }
222:
223:     @GetMapping("/users")
224:     public ResponseEntity<Map<String, Object>> getAllUsers() {
225:         List<User> users = userRepository.findAll();
226:
227:         List<Map<String, Object>> userList = users.stream()
228:                 .map(user -> {
229:                     Map<String, Object> userMap = new HashMap<>();
230:                     userMap.put("id", user.getId());
231:                     userMap.put("name", user.getName());
232:                     userMap.put("email", user.getEmail());
233:                     userMap.put("role", user.getRole().name());
234:                     userMap.put("createdAt", user.getCreatedAt());
235:                     return userMap;
236:                 })
237:                 .collect(Collectors.toList());
238:
239:         Map<String, Object> response = new HashMap<>();
240:         response.put("success", true);
241:         response.put("message", "Users retrieved successfully");
242:         response.put("users", userList);
243:         response.put("count", userList.size());
244:
245:         return ResponseEntity.ok(response);
246:     }
247:
248:     @GetMapping("/feedback")
249:     public ResponseEntity<Map<String, Object>> getAllFeedback() {
250:         List<FeedbackResponse> feedback = feedbackService.getAllFeedback();
251:
252:         Map<String, Object> response = new HashMap<>();
253:         response.put("success", true);
254:         response.put("message", "Feedback retrieved successfully");
255:         response.put("feedback", feedback);
256:         response.put("count", feedback.size());
257:
258:         return ResponseEntity.ok(response);
259:     }
260: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\controller\AuthController.java

```
  1: package com.lostandfound.controller;
  2:
  3: import com.lostandfound.dto.request.LoginRequest;
  4: import com.lostandfound.dto.request.RegisterRequest;
  5: import com.lostandfound.dto.request.TokenRefreshRequest;
  6: import com.lostandfound.dto.response.ApiResponse;
  7: import com.lostandfound.dto.response.AuthResponse;
  8: import com.lostandfound.dto.response.TokenRefreshResponse;
  9: import com.lostandfound.exception.BadRequestException;
 10: import com.lostandfound.security.UserPrincipal;
 11: import com.lostandfound.service.RefreshTokenService;
 12: import com.lostandfound.service.UserService;
 13: import jakarta.servlet.http.HttpServletRequest;
 14: import jakarta.validation.Valid;
 15: import lombok.RequiredArgsConstructor;
 16: import org.springframework.http.ResponseEntity;
 17: import org.springframework.security.core.annotation.AuthenticationPrincipal;
```

```
18: import org.springframework.web.bind.annotation.*;
19:
20: @RestController
21: @RequestMapping("/api/auth")
22: @RequiredArgsConstructor
23: public class AuthController {
24:
25:     private final UserService userService;
26:     private final RefreshTokenService refreshTokenService;
27:
28:     @PostMapping("/register")
29:     public ResponseEntity<AuthResponse> registerUser(
30:             @Valid @RequestBody RegisterRequest request,
31:             HttpServletRequest httpRequest) {
32:
33:         String ipAddress = getClientIP(httpRequest);
34:         String userAgent = httpRequest.getHeader("User-Agent");
35:
36:         AuthResponse response = userService.registerUser(request, ipAddress, userAgent);
37:         return ResponseEntity.ok(response);
38:     }
39:
40:     @PostMapping("/login")
41:     public ResponseEntity<AuthResponse> loginUser(
42:             @Valid @RequestBody LoginRequest request,
43:             HttpServletRequest httpRequest) {
44:
45:         String ipAddress = getClientIP(httpRequest);
46:         String userAgent = httpRequest.getHeader("User-Agent");
47:
48:         AuthResponse response = userService.loginUser(request, ipAddress, userAgent);
49:         return ResponseEntity.ok(response);
50:     }
51:
52:     @PostMapping("/refresh")
53:     public ResponseEntity<TokenRefreshResponse> refreshToken(
54:             @Valid @RequestBody TokenRefreshRequest request,
55:             @AuthenticationPrincipal UserPrincipal currentUser) {
56:
57:         // Validate that refresh token is provided
58:         if (request.getRefreshToken() == null || request.getRefreshToken().trim().isEmpty(
   | )) {
59:             throw new BadRequestException("Refresh token is required");
60:         }
61:
62:         TokenRefreshResponse response = userService.refreshToken(request.getRefreshToken()
   | );
63:         return ResponseEntity.ok(response);
64:     }
65:
66:     @PostMapping("/logout")
67:     public ResponseEntity<ApiResponse> logout(
68:             @AuthenticationPrincipal UserPrincipal currentUser,
69:             @RequestBody(required = false) TokenRefreshRequest request) {
70:
71:         // User must be authenticated to logout
72:         if (currentUser == null) {
73:             throw new BadRequestException("You are not logged in");
74:         }
75:
76:         // Validate and revoke refresh token if provided
77:         if (request != null && request.getRefreshToken() != null && !request.getRefreshTok
   | en().trim().isEmpty()) {
78:             // Verify the token belongs to the current user before revoking
79:             refreshTokenService.revokeTokenForUser(request.getRefreshToken(), currentUser.
   | getId());
80:         } else {
81:             // Revoke all tokens for this user
82:             refreshTokenService.revokeAllUserTokens(currentUser.getId());
83:         }
84:
85:         ApiResponse response = ApiResponse.builder()
86:                 .success(true)
87:                 .message("Logged out successfully")
88:                 .build();
89:
90:         return ResponseEntity.ok(response);
91:     }
92:
93:     @GetMapping("/validate")
94:     public ResponseEntity<ApiResponse> validateToken(
95:             @AuthenticationPrincipal UserPrincipal currentUser) {
96:
97:         if (currentUser == null) {
98:             throw new BadRequestException("Invalid or expired token");
```

```
 99:         }
100:
101:         ApiResponse response = ApiResponse.builder()
102:                 .success(true)
103:                 .message("Token is valid")
104:                 .data(currentUser.getEmail())
105:                 .build();
106:
107:         return ResponseEntity.ok(response);
108:     }
109:
110:     @GetMapping("/me")
111:     public ResponseEntity<ApiResponse> getCurrentUser(
112:             @AuthenticationPrincipal UserPrincipal currentUser) {
113:
114:         if (currentUser == null) {
115:             throw new BadRequestException("Not authenticated");
116:         }
117:
118:         AuthResponse.UserDTO userDTO = AuthResponse.UserDTO.builder()
119:                 .id(currentUser.getId())
120:                 .name(currentUser.getName())
121:                 .email(currentUser.getEmail())
122:                 .role(currentUser.getAuthorities().iterator().next().getAuthority().replac
    | e("ROLE_", ""))
123:                 .build();
124:
125:         ApiResponse response = ApiResponse.builder()
126:                 .success(true)
127:                 .message("User details retrieved successfully")
128:                 .data(userDTO)
129:                 .build();
130:
131:         return ResponseEntity.ok(response);
132:     }
133:
134:     private String getClientIP(HttpServletRequest request) {
135:         String xfHeader = request.getHeader("X-Forwarded-For");
136:         String remoteAddr = request.getRemoteAddr();
137:         boolean isTrustedProxy = isTrustedProxy(remoteAddr);
138:
139:         if (isTrustedProxy && xfHeader != null && !xfHeader.isEmpty()) {
140:             String clientIp = xfHeader.split(",")[0].trim();
141:             if (isValidIP(clientIp)) {
142:                 return clientIp;
143:             }
144:         }
145:
146:         return remoteAddr;
147:     }
148:
149:     private boolean isTrustedProxy(String ip) {
150:         return "127.0.0.1".equals(ip) ||
151:                 "0:0:0:0:0:0:0:1".equals(ip) ||
152:                 "::1".equals(ip);
153:     }
154:
155:     private boolean isValidIP(String ip) {
156:         if (ip == null || ip.isEmpty()) {
157:             return false;
158:         }
159:         String ipv4Pattern = "^((25[0-5]|(2[0-4]|1\\d|[1-9]|)\\d)\\.?\\b){4}$";
160:         return ip.matches(ipv4Pattern);
161:     }
162: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\controller\ClaimController.java

```
  1: package com.lostandfound.controller;
  2:
  3: import com.lostandfound.dto.response.ApiResponse;
  4: import com.lostandfound.dto.response.ClaimResponse;
  5: import com.lostandfound.exception.BadRequestException;
  6: import com.lostandfound.security.UserPrincipal;
  7: import com.lostandfound.service.ClaimService;
  8: import lombok.RequiredArgsConstructor;
  9: import org.springframework.http.ResponseEntity;
 10: import org.springframework.security.core.annotation.AuthenticationPrincipal;
 11: import org.springframework.web.bind.annotation.*;
 12:
 13: import java.util.HashMap;
 14: import java.util.List;
 15: import java.util.Map;
```

```
16:
17: @RestController
18: @RequestMapping("/claims")
19: @RequiredArgsConstructor
20: public class ClaimController {
21:
22:     private final ClaimService claimService;
23:
24:     @PostMapping("/item/{itemId}")
25:     public ResponseEntity<ApiResponse> claimItem(
26:             @PathVariable Long itemId,
27:             @AuthenticationPrincipal UserPrincipal currentUser) {
28:
29:         if (currentUser == null) {
30:             throw new BadRequestException("You must be logged in to claim an item");
31:         }
32:
33:         ApiResponse response = claimService.claimItem(itemId, currentUser);
34:         return ResponseEntity.ok(response);
35:     }
36:
37:     @GetMapping
38:     public ResponseEntity<Map<String, Object>> getMyClaims(
39:             @AuthenticationPrincipal UserPrincipal currentUser) {
40:
41:         if (currentUser == null) {
42:             throw new BadRequestException("You must be logged in to view your claims");
43:         }
44:
45:         List<ClaimResponse> claims = claimService.getUserClaims(currentUser);
46:
47:         Map<String, Object> response = new HashMap<>();
48:         response.put("success", true);
49:         response.put("message", "Claims retrieved successfully");
50:         response.put("claims", claims);
51:         response.put("count", claims.size());
52:
53:         return ResponseEntity.ok(response);
54:     }
55:
56:     @DeleteMapping("/{claimId}")
57:     public ResponseEntity<ApiResponse> deleteClaim(
58:             @PathVariable Long claimId,
59:             @AuthenticationPrincipal UserPrincipal currentUser) {
60:
61:         if (currentUser == null) {
62:             throw new BadRequestException("You must be logged in to delete a claim");
63:         }
64:
65:         claimService.deleteClaimByUser(claimId, currentUser);
66:
67:         ApiResponse response = ApiResponse.builder()
68:                 .success(true)
69:                 .message("Claim deleted successfully")
70:                 .build();
71:
72:         return ResponseEntity.ok(response);
73:     }
74: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\controller\DashboardController.java

```
  1: package com.lostandfound.controller;
  2:
  3: import com.lostandfound.dto.response.ClaimResponse;
  4: import com.lostandfound.dto.response.DashboardResponse;
  5: import com.lostandfound.dto.response.ItemResponse;
  6: import com.lostandfound.dto.response.MessageResponse;
  7: import com.lostandfound.exception.BadRequestException;
  8: import com.lostandfound.security.UserPrincipal;
  9: import com.lostandfound.service.ClaimService;
 10: import com.lostandfound.service.ItemService;
 11: import com.lostandfound.service.MessageService;
 12: import lombok.RequiredArgsConstructor;
 13: import org.springframework.http.ResponseEntity;
 14: import org.springframework.security.core.annotation.AuthenticationPrincipal;
 15: import org.springframework.web.bind.annotation.GetMapping;
 16: import org.springframework.web.bind.annotation.RequestMapping;
 17: import org.springframework.web.bind.annotation.RestController;
 18:
 19: import java.util.List;
 20:
 21: @RestController
```

```
22: @RequestMapping("/dashboard")
23: @RequiredArgsConstructor
24: public class DashboardController {
25:
26:     private final ItemService itemService;
27:     private final ClaimService claimService;
28:     private final MessageService messageService;
29:
30:     @GetMapping
31:     public ResponseEntity<DashboardResponse> getDashboard(
32:             @AuthenticationPrincipal UserPrincipal currentUser) {
33:
34:         // Validate user is authenticated
35:         if (currentUser == null) {
36:             throw new BadRequestException("You must be logged in to access the dashboard")
   | ;
37:         }
38:
39:         List<ItemResponse> items = itemService.getUserItems(currentUser);
40:         List<ClaimResponse> claims = claimService.getUserClaims(currentUser);
41:         List<MessageResponse> messages = messageService.getUserMessages(currentUser);
42:
43:         String role = currentUser.getAuthorities().iterator().next()
44:                 .getAuthority().replace("ROLE_", "");
45:
46:         DashboardResponse.UserInfo userInfo = DashboardResponse.UserInfo.builder()
47:                 .name(currentUser.getName())
48:                 .email(currentUser.getEmail())
49:                 .role(role)
50:                 .build();
51:
52:         DashboardResponse response = DashboardResponse.builder()
53:                 .user(userInfo)
54:                 .items(items)
55:                 .claims(claims)
56:                 .messages(messages)
57:                 .build();
58:
59:         return ResponseEntity.ok(response);
60:     }
61: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\controller\FeedbackController.java

```
  1: package com.lostandfound.controller;
  2:
  3: import com.lostandfound.dto.request.FeedbackRequest;
  4: import com.lostandfound.dto.response.ApiResponse;
  5: import com.lostandfound.exception.BadRequestException;
  6: import com.lostandfound.security.UserPrincipal;
  7: import com.lostandfound.service.FeedbackService;
  8: import jakarta.validation.Valid;
  9: import lombok.RequiredArgsConstructor;
 10: import org.springframework.http.ResponseEntity;
 11: import org.springframework.security.core.annotation.AuthenticationPrincipal;
 12: import org.springframework.web.bind.annotation.*;
 13:
 14: @RestController
 15: @RequestMapping("/feedback")
 16: @RequiredArgsConstructor
 17: public class FeedbackController {
 18:
 19:     private final FeedbackService feedbackService;
 20:
 21:     @PostMapping
 22:     public ResponseEntity<ApiResponse> submitFeedback(
 23:             @Valid @RequestBody FeedbackRequest request,
 24:             @AuthenticationPrincipal UserPrincipal currentUser) {
 25:
 26:         // Validate user is authenticated
 27:         if (currentUser == null) {
 28:             throw new BadRequestException("You must be logged in to submit feedback");
 29:         }
 30:
 31:         // Additional validation for feedback content
 32:         if (request.getFeedback() == null || request.getFeedback().trim().isEmpty()) {
 33:             throw new BadRequestException("Feedback content cannot be empty");
 34:         }
 35:
 36:         if (request.getFeedback().trim().length() < 10) {
 37:             throw new BadRequestException("Feedback must be at least 10 characters long");
 38:         }
 39:
```

```
 40:         if (request.getFeedback().trim().length() > 2000) {
 41:             throw new BadRequestException("Feedback cannot exceed 2000 characters");
 42:         }
 43:
 44:         ApiResponse response = feedbackService.submitFeedback(request, currentUser);
 45:         return ResponseEntity.ok(response);
 46:     }
 47: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\controller\ItemController.java

```
  1: package com.lostandfound.controller;
  2:
  3: import com.lostandfound.dto.request.ItemRequest;
  4: import com.lostandfound.dto.response.ApiResponse;
  5: import com.lostandfound.dto.response.ItemResponse;
  6: import com.lostandfound.exception.BadRequestException;
  7: import com.lostandfound.security.UserPrincipal;
  8: import com.lostandfound.service.ItemService;
  9: import jakarta.validation.Valid;
 10: import lombok.RequiredArgsConstructor;
 11: import org.springframework.http.ResponseEntity;
 12: import org.springframework.security.core.annotation.AuthenticationPrincipal;
 13: import org.springframework.web.bind.annotation.*;
 14: import org.springframework.web.multipart.MultipartFile;
 15:
 16: import java.util.HashMap;
 17: import java.util.List;
 18: import java.util.Map;
 19:
 20: @RestController
 21: @RequestMapping("/items")
 22: @RequiredArgsConstructor
 23: public class ItemController {
 24:
 25:     private final ItemService itemService;
 26:
 27:     @PostMapping
 28:     public ResponseEntity<ApiResponse> createItem(
 29:             @Valid @ModelAttribute ItemRequest request,
 30:             @RequestParam(value = "image", required = false) MultipartFile image,
 31:             @AuthenticationPrincipal UserPrincipal currentUser) {
 32:
 33:         // Validate user is authenticated
 34:         if (currentUser == null) {
 35:             throw new BadRequestException("You must be logged in to create an item");
 36:         }
 37:
 38:         // Validate image if provided
 39:         if (image != null && !image.isEmpty()) {
 40:             // Check file size (5MB max)
 41:             if (image.getSize() > 5 * 1024 * 1024) {
 42:                 throw new BadRequestException("Image size must not exceed 5MB");
 43:             }
 44:
 45:             // Check file type
 46:             String contentType = image.getContentType();
 47:             if (contentType == null || !contentType.startsWith("image/")) {
 48:                 throw new BadRequestException("Only image files are allowed");
 49:             }
 50:         }
 51:
 52:         ItemResponse itemResponse = itemService.createItem(request, image, currentUser);
 53:
 54:         ApiResponse response = ApiResponse.builder()
 55:                 .success(true)
 56:                 .message("Item registered successfully")
 57:                 .data(itemResponse)
 58:                 .build();
 59:
 60:         return ResponseEntity.ok(response);
 61:     }
 62:
 63:     @GetMapping
 64:     public ResponseEntity<Map<String, Object>> getItems(
 65:             @RequestParam(required = false, defaultValue = "") String search,
 66:             @RequestParam(required = false, defaultValue = "") String status) {
 67:
 68:         List<ItemResponse> items = itemService.searchItems(search, status);
 69:
 70:         Map<String, Object> response = new HashMap<>();
 71:         response.put("success", true);
 72:         response.put("message", "Items retrieved successfully");
```

```
 73:         response.put("items", items);
 74:         response.put("count", items.size());
 75:
 76:         return ResponseEntity.ok(response);
 77:     }
 78:
 79:     @GetMapping("/{itemId}")
 80:     public ResponseEntity<ApiResponse> getItemById(@PathVariable Long itemId) {
 81:         ItemResponse item = itemService.getItemById(itemId);
 82:
 83:         ApiResponse response = ApiResponse.builder()
 84:                 .success(true)
 85:                 .message("Item retrieved successfully")
 86:                 .data(item)
 87:                 .build();
 88:
 89:         return ResponseEntity.ok(response);
 90:     }
 91:
 92:     @PutMapping("/{itemId}")
 93:     public ResponseEntity<ApiResponse> updateItem(
 94:             @PathVariable Long itemId,
 95:             @Valid @ModelAttribute ItemRequest request,
 96:             @RequestParam(value = "image", required = false) MultipartFile image,
 97:             @AuthenticationPrincipal UserPrincipal currentUser) {
 98:
 99:         // Validate user is authenticated
100:         if (currentUser == null) {
101:             throw new BadRequestException("You must be logged in to update an item");
102:         }
103:
104:         // Validate image if provided
105:         if (image != null && !image.isEmpty()) {
106:             if (image.getSize() > 5 * 1024 * 1024) {
107:                 throw new BadRequestException("Image size must not exceed 5MB");
108:             }
109:
110:             String contentType = image.getContentType();
111:             if (contentType == null || !contentType.startsWith("image/")) {
112:                 throw new BadRequestException("Only image files are allowed");
113:             }
114:         }
115:
116:         ItemResponse itemResponse = itemService.updateItem(itemId, request, image, current
    | User);
117:
118:         ApiResponse response = ApiResponse.builder()
119:                 .success(true)
120:                 .message("Item updated successfully")
121:                 .data(itemResponse)
122:                 .build();
123:
124:         return ResponseEntity.ok(response);
125:     }
126:
127:     @DeleteMapping("/{itemId}")
128:     public ResponseEntity<ApiResponse> deleteItem(
129:             @PathVariable Long itemId,
130:             @AuthenticationPrincipal UserPrincipal currentUser) {
131:
132:         // Validate user is authenticated
133:         if (currentUser == null) {
134:             throw new BadRequestException("You must be logged in to delete an item");
135:         }
136:
137:         itemService.deleteItem(itemId, currentUser);
138:
139:         ApiResponse response = ApiResponse.builder()
140:                 .success(true)
141:                 .message("Item deleted successfully")
142:                 .build();
143:
144:         return ResponseEntity.ok(response);
145:     }
146: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\controller\MessageController.java

```
  1: package com.lostandfound.controller;
  2:
  3: import com.lostandfound.dto.request.MessageRequest;
  4: import com.lostandfound.dto.response.ApiResponse;
  5: import com.lostandfound.dto.response.MessageResponse;
```

```java
import com.lostandfound.exception.BadRequestException;
import com.lostandfound.security.UserPrincipal;
import com.lostandfound.service.MessageService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.*;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

@RestController
@RequestMapping("/messages")
@RequiredArgsConstructor
public class MessageController {

    private final MessageService messageService;

    @PostMapping
    public ResponseEntity<ApiResponse> sendMessage(
            @Valid @RequestBody MessageRequest request,
            @AuthenticationPrincipal UserPrincipal currentUser) {

        // Validate user is authenticated
        if (currentUser == null) {
            throw new BadRequestException("You must be logged in to send a message");
        }

        // Validate user is not sending message to themselves
        if (request.getReceiverId().equals(currentUser.getId())) {
            throw new BadRequestException("You cannot send a message to yourself");
        }

        ApiResponse response = messageService.sendMessage(request, currentUser);
        return ResponseEntity.ok(response);
    }

    @PostMapping("/reply")
    public ResponseEntity<ApiResponse> replyMessage(
            @Valid @RequestBody MessageRequest request,
            @AuthenticationPrincipal UserPrincipal currentUser) {

        // Validate user is authenticated
        if (currentUser == null) {
            throw new BadRequestException("You must be logged in to send a reply");
        }

        // Validate user is not replying to themselves
        if (request.getReceiverId().equals(currentUser.getId())) {
            throw new BadRequestException("You cannot send a message to yourself");
        }

        ApiResponse response = messageService.sendMessage(request, currentUser);
        return ResponseEntity.ok(response);
    }

    @GetMapping
    public ResponseEntity<Map<String, Object>> getMyMessages(
            @AuthenticationPrincipal UserPrincipal currentUser) {

        // Validate user is authenticated
        if (currentUser == null) {
            throw new BadRequestException("You must be logged in to view messages");
        }

        List<MessageResponse> messages = messageService.getUserMessages(currentUser);

        Map<String, Object> response = new HashMap<>();
        response.put("success", true);
        response.put("message", "Messages retrieved successfully");
        response.put("messages", messages);
        response.put("count", messages.size());

        return ResponseEntity.ok(response);
    }

    @GetMapping("/sent")
    public ResponseEntity<Map<String, Object>> getSentMessages(
            @AuthenticationPrincipal UserPrincipal currentUser) {

        // Validate user is authenticated
        if (currentUser == null) {
            throw new BadRequestException("You must be logged in to view sent messages");
```

```
 91:         }
 92:
 93:         List<MessageResponse> messages = messageService.getSentMessages(currentUser);
 94:
 95:         Map<String, Object> response = new HashMap<>();
 96:         response.put("success", true);
 97:         response.put("message", "Sent messages retrieved successfully");
 98:         response.put("messages", messages);
 99:         response.put("count", messages.size());
100:
101:         return ResponseEntity.ok(response);
102:     }
103:
104:     @DeleteMapping("/{messageId}")
105:     public ResponseEntity<ApiResponse> deleteMessage(
106:             @PathVariable Long messageId,
107:             @AuthenticationPrincipal UserPrincipal currentUser) {
108:
109:         // Validate user is authenticated
110:         if (currentUser == null) {
111:             throw new BadRequestException("You must be logged in to delete a message");
112:         }
113:
114:         messageService.deleteMessage(messageId, currentUser);
115:
116:         ApiResponse response = ApiResponse.builder()
117:                 .success(true)
118:                 .message("Message deleted successfully")
119:                 .build();
120:
121:         return ResponseEntity.ok(response);
122:     }
123: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\controller\RateLimitController.java

```
  1: package com.lostandfound.controller;
  2:
  3: import com.lostandfound.config.RateLimitConfig;
  4: import com.lostandfound.dto.response.ApiResponse;
  5: import lombok.RequiredArgsConstructor;
  6: import org.springframework.http.ResponseEntity;
  7: import org.springframework.security.access.prepost.PreAuthorize;
  8: import org.springframework.web.bind.annotation.*;
  9:
 10: import java.util.HashMap;
 11: import java.util.Map;
 12:
 13: @RestController
 14: @RequestMapping("/admin/rate-limit")
 15: @PreAuthorize("hasRole('ADMIN')")
 16: @RequiredArgsConstructor
 17: public class RateLimitController {
 18:
 19:     private final RateLimitConfig rateLimitConfig;
 20:
 21:     /**
 22:      * Get rate limit statistics
 23:      */
 24:     @GetMapping("/stats")
 25:     public ResponseEntity<ApiResponse> getRateLimitStats() {
 26:         Map<String, Object> stats = new HashMap<>();
 27:         stats.put("activeBuckets", rateLimitConfig.getCacheSize());
 28:         stats.put("rateLimitTypes", getRateLimitInfo());
 29:
 30:         ApiResponse response = ApiResponse.builder()
 31:                 .success(true)
 32:                 .message("Rate limit statistics")
 33:                 .data(stats)
 34:                 .build();
 35:
 36:         return ResponseEntity.ok(response);
 37:     }
 38:
 39:     /**
 40:      * Clear rate limit for specific IP
 41:      */
 42:     @DeleteMapping("/clear/{ip}")
 43:     public ResponseEntity<ApiResponse> clearRateLimitForIp(@PathVariable String ip) {
 44:         // Clear all buckets for this IP across all limit types
 45:         for (RateLimitConfig.RateLimitType type : RateLimitConfig.RateLimitType.values())
    | {
 46:             String key = ip + ":" + type.name();
```

```
 47:             rateLimitConfig.clearBucket(key);
 48:         }
 49:
 50:         ApiResponse response = ApiResponse.builder()
 51:                 .success(true)
 52:                 .message("Rate limit cleared for IP: " + ip)
 53:                 .build();
 54:
 55:         return ResponseEntity.ok(response);
 56:     }
 57:
 58:     /**
 59:      * Clear all rate limits (use with caution)
 60:      */
 61:     @DeleteMapping("/clear-all")
 62:     public ResponseEntity<ApiResponse> clearAllRateLimits() {
 63:         rateLimitConfig.clearAllBuckets();
 64:
 65:         ApiResponse response = ApiResponse.builder()
 66:                 .success(true)
 67:                 .message("All rate limits cleared")
 68:                 .build();
 69:
 70:         return ResponseEntity.ok(response);
 71:     }
 72:
 73:     /**
 74:      * Get rate limit configuration info
 75:      */
 76:     private Map<String, Object> getRateLimitInfo() {
 77:         Map<String, Object> info = new HashMap<>();
 78:
 79:         for (RateLimitConfig.RateLimitType type : RateLimitConfig.RateLimitType.values())
    | {
 80:             Map<String, Object> typeInfo = new HashMap<>();
 81:             typeInfo.put("capacity", type.getCapacity());
 82:             typeInfo.put("refillDurationSeconds", type.getRefillDuration().getSeconds());
 83:             info.put(type.name(), typeInfo);
 84:         }
 85:
 86:         return info;
 87:     }
 88: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\dto\request\FeedbackRequest.java

```
  1: package com.lostandfound.dto.request;
  2:
  3: import jakarta.validation.constraints.NotBlank;
  4: import lombok.Data;
  5:
  6: @Data
  7: public class FeedbackRequest {
  8:
  9:     @NotBlank(message = "Feedback text is required")
 10:     private String feedback;
 11: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\dto\request\ItemRequest.java

```
  1: package com.lostandfound.dto.request;
  2:
  3: import jakarta.validation.constraints.NotBlank;
  4: import lombok.Data;
  5:
  6: @Data
  7: public class ItemRequest {
  8:
  9:     @NotBlank(message = "Item name is required")
 10:     private String name;
 11:
 12:     @NotBlank(message = "Description is required")
 13:     private String description;
 14:
 15:     @NotBlank(message = "Location is required")
 16:     private String location;
 17:
 18:     @NotBlank(message = "Status is required")
 19:     private String status;
 20: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\dto\request\LoginRequest.java

```
  1: package com.lostandfound.dto.request;
  2:
  3: import jakarta.validation.constraints.Email;
  4: import jakarta.validation.constraints.NotBlank;
  5: import jakarta.validation.constraints.Size;
  6: import lombok.Data;
  7:
  8: @Data
  9: public class LoginRequest {
 10:
 11:     @NotBlank(message = "Email is required")
 12:     @Email(message = "Please provide a valid email address")
 13:     @Size(max = 255, message = "Email must not exceed 255 characters")
 14:     private String email;
 15:
 16:     @NotBlank(message = "Password is required")
 17:     @Size(min = 8, max = 128, message = "Password must be between 8 and 128 characters")
 18:     private String password;
 19: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\dto\request\MessageRequest.java

```
  1: package com.lostandfound.dto.request;
  2:
  3: import jakarta.validation.constraints.NotBlank;
  4: import jakarta.validation.constraints.NotNull;
  5: import lombok.Data;
  6:
  7: @Data
  8: public class MessageRequest {
  9:
 10:     @NotNull(message = "Receiver ID is required")
 11:     private Long receiverId;
 12:
 13:     @NotNull(message = "Item ID is required")
 14:     private Long itemId;
 15:
 16:     @NotBlank(message = "Message is required")
 17:     private String message;
 18: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\dto\request\RegisterRequest.java

```
  1: package com.lostandfound.dto.request;
  2:
  3: import jakarta.validation.constraints.Email;
  4: import jakarta.validation.constraints.NotBlank;
  5: import jakarta.validation.constraints.Pattern;
  6: import jakarta.validation.constraints.Size;
  7: import lombok.Data;
  8: import lombok.AllArgsConstructor;
  9: import lombok.NoArgsConstructor;
 10:
 11: @Data
 12: @NoArgsConstructor
 13: @AllArgsConstructor
 14: public class RegisterRequest {
 15:
 16:     @NotBlank(message = "Name is required")
 17:     @Size(min = 2, max = 100, message = "Name must be between 2 and 100 characters")
 18:     @Pattern(regexp = "^[a-zA-Z\\s]+$", message = "Name should only contain letters and sp
    | aces")
 19:     private String name;
 20:
 21:     @NotBlank(message = "Email is required")
 22:     @Email(message = "Please provide a valid email address")
 23:     @Size(max = 255, message = "Email must not exceed 255 characters")
 24:     private String email;
 25:
 26:     @NotBlank(message = "Password is required")
 27:     @Size(min = 8, max = 128, message = "Password must be between 8 and 128 characters")
 28:     @Pattern(
 29:             regexp = "^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d)(?=.*[^a-zA-Z\\d]).{8,}$",
 30:             message = "Password must contain at least one uppercase letter, one lowercase
    | letter, one number, and one special character"
 31:     )
 32:     private String password;
 33:
 34:     @NotBlank(message = "Confirm password is required")
```

```
 35:     private String confirmPassword;
 36: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\dto\request\TokenRefreshRequest.java

```
  1: package com.lostandfound.dto.request;
  2:
  3: import jakarta.validation.constraints.NotBlank;
  4: import lombok.AllArgsConstructor;
  5: import lombok.Data;
  6: import lombok.NoArgsConstructor;
  7:
  8: @Data
  9: @NoArgsConstructor
 10: @AllArgsConstructor
 11: public class TokenRefreshRequest {
 12:
 13:     @NotBlank(message = "Refresh token is required")
 14:     private String refreshToken;
 15: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\dto\response\ApiResponse.java

```
  1: package com.lostandfound.dto.response;
  2:
  3: import lombok.AllArgsConstructor;
  4: import lombok.Builder;
  5: import lombok.Data;
  6: import lombok.NoArgsConstructor;
  7:
  8: @Data
  9: @Builder
 10: @NoArgsConstructor
 11: @AllArgsConstructor
 12: public class ApiResponse {
 13:     private boolean success;
 14:     private String message;
 15:     private Object data;
 16: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\dto\response\AuthResponse.java

```
  1: package com.lostandfound.dto.response;
  2:
  3: import lombok.AllArgsConstructor;
  4: import lombok.Builder;
  5: import lombok.Data;
  6: import lombok.NoArgsConstructor;
  7:
  8: @Data
  9: @Builder
 10: @NoArgsConstructor
 11: @AllArgsConstructor
 12: public class AuthResponse {
 13:     private boolean success;
 14:     private String message;
 15:     private String accessToken;
 16:     private String refreshToken;
 17:     @Builder.Default
 18:     private String tokenType = "Bearer";
 19:     private Long expiresIn; // in seconds
 20:     private UserDTO user;
 21:
 22:     @Data
 23:     @Builder
 24:     @NoArgsConstructor
 25:     @AllArgsConstructor
 26:     public static class UserDTO {
 27:         private Long id;
 28:         private String name;
 29:         private String email;
 30:         private String role;
 31:     }
 32: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\dto\response\ClaimResponse.java

```
  1: package com.lostandfound.dto.response;
  2:
  3: import java.time.LocalDateTime;
  4:
  5: import lombok.AllArgsConstructor;
  6: import lombok.Builder;
  7: import lombok.Data;
  8: import lombok.NoArgsConstructor;
  9:
 10: @Data
 11: @Builder
 12: @NoArgsConstructor
 13: @AllArgsConstructor
 14: public class ClaimResponse {
 15:     private Long id;
 16:     private Long itemId;
 17:     private String itemName;
 18:     private String description;
 19:     private String location;
 20:     private Long claimedBy;
 21:     private String claimerName;
 22:     private String claimantName;
 23:     private String claimantEmail;
 24:     private LocalDateTime claimedAt;
 25: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\dto\response\DashboardResponse.java

```
  1: package com.lostandfound.dto.response;
  2:
  3: import java.util.List;
  4:
  5: import lombok.AllArgsConstructor;
  6: import lombok.Builder;
  7: import lombok.Data;
  8: import lombok.NoArgsConstructor;
  9:
 10: @Data
 11: @Builder
 12: @NoArgsConstructor
 13: @AllArgsConstructor
 14: public class DashboardResponse {
 15:     private UserInfo user;
 16:     private List<ItemResponse> items;
 17:     private List<ClaimResponse> claims;
 18:     private List<MessageResponse> messages;
 19:
 20:     @Data
 21:     @Builder
 22:     @NoArgsConstructor
 23:     @AllArgsConstructor
 24:     public static class UserInfo {
 25:         private String name;
 26:         private String email;
 27:         private String role;
 28:     }
 29: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\dto\response\FeedbackResponse.java

```
  1: package com.lostandfound.dto.response;
  2:
  3: import java.time.LocalDateTime;
  4:
  5: import lombok.AllArgsConstructor;
  6: import lombok.Builder;
  7: import lombok.Data;
  8: import lombok.NoArgsConstructor;
  9:
 10: @Data
 11: @Builder
 12: @NoArgsConstructor
 13: @AllArgsConstructor
 14: public class FeedbackResponse {
 15:     private Long id;
 16:     private Long userId;
 17:     private String userName;
 18:     private String feedbackText;
 19:     private LocalDateTime submittedAt;
```

```
 20: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\dto\response\ItemResponse.java

```
  1: package com.lostandfound.dto.response;
  2:
  3: import java.time.LocalDateTime;
  4:
  5: import lombok.AllArgsConstructor;
  6: import lombok.Builder;
  7: import lombok.Data;
  8: import lombok.NoArgsConstructor;
  9:
 10: @Data
 11: @Builder
 12: @NoArgsConstructor
 13: @AllArgsConstructor
 14: public class ItemResponse {
 15:     private Long id;
 16:     private String name;
 17:     private String description;
 18:     private String location;
 19:     private String status;
 20:     private String image;
 21:     private Long createdBy;
 22:     private String creatorName;
 23:     private LocalDateTime createdAt;
 24: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\dto\response\MessageResponse.java

```
  1: package com.lostandfound.dto.response;
  2:
  3: import java.time.LocalDateTime;
  4:
  5: import lombok.AllArgsConstructor;
  6: import lombok.Builder;
  7: import lombok.Data;
  8: import lombok.NoArgsConstructor;
  9:
 10: @Data
 11: @Builder
 12: @NoArgsConstructor
 13: @AllArgsConstructor
 14: public class MessageResponse {
 15:     private Long id;
 16:     private Long senderId;
 17:     private String senderName;
 18:     private Long receiverId;
 19:     private Long itemId;
 20:     private String itemName;
 21:     private String message;
 22:     private LocalDateTime sentAt;
 23: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\dto\response\TokenRefreshResponse.java

```
  1: package com.lostandfound.dto.response;
  2:
  3: import lombok.AllArgsConstructor;
  4: import lombok.Builder;
  5: import lombok.Data;
  6: import lombok.NoArgsConstructor;
  7:
  8: @Data
  9: @Builder
 10: @NoArgsConstructor
 11: @AllArgsConstructor
 12: public class TokenRefreshResponse {
 13:     private boolean success;
 14:     private String message;
 15:     private String accessToken;
 16:     private String refreshToken;
 17:     @Builder.Default
 18:     private String tokenType = "Bearer";
 19: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\exception\BadRequestException.java

```
  1: package com.lostandfound.exception;
  2:
  3: import org.springframework.http.HttpStatus;
  4: import org.springframework.web.bind.annotation.ResponseStatus;
  5:
  6: @ResponseStatus(HttpStatus.BAD_REQUEST)
  7: public class BadRequestException extends RuntimeException {
  8:
  9:     public BadRequestException(String message) {
 10:         super(message);
 11:     }
 12: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\exception\GlobalExceptionHandler.java

```
  1: package com.lostandfound.exception;
  2:
  3: import com.lostandfound.dto.response.ApiResponse;
  4: import org.slf4j.Logger;
  5: import org.slf4j.LoggerFactory;
  6: import org.springframework.http.HttpStatus;
  7: import org.springframework.http.ResponseEntity;
  8: import org.springframework.security.authentication.BadCredentialsException;
  9: import org.springframework.security.core.userdetails.UsernameNotFoundException;
 10: import org.springframework.validation.FieldError;
 11: import org.springframework.web.bind.MethodArgumentNotValidException;
 12: import org.springframework.web.bind.annotation.ExceptionHandler;
 13: import org.springframework.web.bind.annotation.RestControllerAdvice;
 14: import org.springframework.web.context.request.WebRequest;
 15: import org.springframework.web.multipart.MaxUploadSizeExceededException;
 16:
 17: import java.util.HashMap;
 18: import java.util.Map;
 19:
 20: @RestControllerAdvice
 21: public class GlobalExceptionHandler {
 22:
 23:     private static final Logger logger = LoggerFactory.getLogger(GlobalExceptionHandler.cl
    | ass);
 24:
 25:     @ExceptionHandler(ResourceNotFoundException.class)
 26:     public ResponseEntity<ApiResponse> handleResourceNotFoundException(
 27:             ResourceNotFoundException ex, WebRequest request) {
 28:
 29:         logger.warn("Resource not found: {}", ex.getMessage());
 30:
 31:         ApiResponse response = ApiResponse.builder()
 32:             .success(false)
 33:             .message(ex.getMessage())
 34:             .build();
 35:         return new ResponseEntity<>(response, HttpStatus.NOT_FOUND);
 36:     }
 37:
 38:     @ExceptionHandler(BadRequestException.class)
 39:     public ResponseEntity<ApiResponse> handleBadRequestException(
 40:             BadRequestException ex, WebRequest request) {
 41:
 42:         logger.warn("Bad request: {}", ex.getMessage());
 43:
 44:         ApiResponse response = ApiResponse.builder()
 45:             .success(false)
 46:             .message(ex.getMessage())
 47:             .build();
 48:         return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
 49:     }
 50:
 51:     @ExceptionHandler(UnauthorizedException.class)
 52:     public ResponseEntity<ApiResponse> handleUnauthorizedException(
 53:             UnauthorizedException ex, WebRequest request) {
 54:
 55:         logger.warn("Unauthorized access: {}", ex.getMessage());
 56:
 57:         ApiResponse response = ApiResponse.builder()
 58:             .success(false)
 59:             .message(ex.getMessage())
 60:             .build();
 61:         return new ResponseEntity<>(response, HttpStatus.UNAUTHORIZED);
 62:     }
 63:
 64:     @ExceptionHandler(BadCredentialsException.class)
 65:     public ResponseEntity<ApiResponse> handleBadCredentialsException(
```

```
 66:             BadCredentialsException ex, WebRequest request) {
 67:
 68:         logger.warn("Invalid credentials attempt");
 69:
 70:         ApiResponse response = ApiResponse.builder()
 71:             .success(false)
 72:             .message("Invalid email or password")
 73:             .build();
 74:         return new ResponseEntity<>(response, HttpStatus.UNAUTHORIZED);
 75:     }
 76:
 77:     @ExceptionHandler(UsernameNotFoundException.class)
 78:     public ResponseEntity<ApiResponse> handleUsernameNotFoundException(
 79:             UsernameNotFoundException ex, WebRequest request) {
 80:
 81:         logger.warn("User not found");
 82:
 83:         // Don't reveal if user exists or not for security
 84:         ApiResponse response = ApiResponse.builder()
 85:             .success(false)
 86:             .message("Invalid email or password")
 87:             .build();
 88:         return new ResponseEntity<>(response, HttpStatus.UNAUTHORIZED);
 89:     }
 90:
 91:     @ExceptionHandler(MethodArgumentNotValidException.class)
 92:     public ResponseEntity<Map<String, Object>> handleValidationExceptions(
 93:             MethodArgumentNotValidException ex) {
 94:
 95:         Map<String, String> errors = new HashMap<>();
 96:         ex.getBindingResult().getAllErrors().forEach((error) -> {
 97:             String fieldName = ((FieldError) error).getField();
 98:             String errorMessage = error.getDefaultMessage();
 99:             errors.put(fieldName, errorMessage);
100:         });
101:
102:         logger.warn("Validation errors: {}", errors);
103:
104:         Map<String, Object> response = new HashMap<>();
105:         response.put("success", false);
106:         response.put("message", "Validation failed");
107:         response.put("errors", errors);
108:
109:         return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
110:     }
111:
112:     @ExceptionHandler(MaxUploadSizeExceededException.class)
113:     public ResponseEntity<ApiResponse> handleMaxUploadSizeExceededException(
114:             MaxUploadSizeExceededException ex) {
115:
116:         logger.warn("File size exceeded");
117:
118:         ApiResponse response = ApiResponse.builder()
119:             .success(false)
120:             .message("File size exceeds maximum allowed size of 5MB")
121:             .build();
122:         return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
123:     }
124:
125:     @ExceptionHandler(Exception.class)
126:     public ResponseEntity<ApiResponse> handleGlobalException(
127:             Exception ex, WebRequest request) {
128:
129:         // Log the full exception for debugging
130:         logger.error("Unexpected error occurred", ex);
131:
132:         // Don't expose internal error details to client
133:         ApiResponse response = ApiResponse.builder()
134:             .success(false)
135:             .message("An unexpected error occurred. Please try again later.")
136:             .build();
137:         return new ResponseEntity<>(response, HttpStatus.INTERNAL_SERVER_ERROR);
138:     }
139: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\exception\ResourceNotFoundException.java

```
  1: package com.lostandfound.exception;
  2:
  3: import org.springframework.http.HttpStatus;
  4: import org.springframework.web.bind.annotation.ResponseStatus;
  5:
  6: @ResponseStatus(HttpStatus.NOT_FOUND)
```

```
 7: public class ResourceNotFoundException extends RuntimeException {
 8:
 9:     public ResourceNotFoundException(String message) {
10:         super(message);
11:     }
12:
13:     public ResourceNotFoundException(String resourceName, String fieldName, Object fieldVa
   | lue) {
14:         super(String.format("%s not found with %s: '%s'", resourceName, fieldName, fieldVa
   | lue));
15:     }
16: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\exception\UnauthorizedException.java

```
 1: package com.lostandfound.exception;
 2:
 3: import org.springframework.http.HttpStatus;
 4: import org.springframework.web.bind.annotation.ResponseStatus;
 5:
 6: @ResponseStatus(HttpStatus.UNAUTHORIZED)
 7: public class UnauthorizedException extends RuntimeException {
 8:
 9:     public UnauthorizedException(String message) {
10:         super(message);
11:     }
12: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\model\Claim.java

```
 1: package com.lostandfound.model;
 2:
 3: import java.time.LocalDateTime;
 4:
 5: import org.hibernate.annotations.CreationTimestamp;
 6:
 7: import jakarta.persistence.Column;
 8: import jakarta.persistence.Entity;
 9: import jakarta.persistence.FetchType;
10: import jakarta.persistence.GeneratedValue;
11: import jakarta.persistence.GenerationType;
12: import jakarta.persistence.Id;
13: import jakarta.persistence.JoinColumn;
14: import jakarta.persistence.ManyToOne;
15: import jakarta.persistence.Table;
16: import lombok.AllArgsConstructor;
17: import lombok.Data;
18: import lombok.NoArgsConstructor;
19:
20: @Entity
21: @Table(name = "claims")
22: @Data
23: @NoArgsConstructor
24: @AllArgsConstructor
25: public class Claim {
26:
27:     @Id
28:     @GeneratedValue(strategy = GenerationType.IDENTITY)
29:     private Long id;
30:
31:     @ManyToOne(fetch = FetchType.LAZY)
32:     @JoinColumn(name = "item_id", nullable = false)
33:     private Item item;
34:
35:     @ManyToOne(fetch = FetchType.LAZY)
36:     @JoinColumn(name = "claimed_by", nullable = false)
37:     private User claimedBy;
38:
39:     @Column(name = "claimant_name", nullable = false)
40:     private String claimantName;
41:
42:     @Column(name = "claimant_email", nullable = false)
43:     private String claimantEmail;
44:
45:     @CreationTimestamp
46:     @Column(name = "claimed_at", nullable = false, updatable = false)
47:     private LocalDateTime claimedAt;
48: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\model\Feedback.java

```
  1: package com.lostandfound.model;
  2:
  3: import java.time.LocalDateTime;
  4:
  5: import org.hibernate.annotations.CreationTimestamp;
  6:
  7: import jakarta.persistence.Column;
  8: import jakarta.persistence.Entity;
  9: import jakarta.persistence.FetchType;
 10: import jakarta.persistence.GeneratedValue;
 11: import jakarta.persistence.GenerationType;
 12: import jakarta.persistence.Id;
 13: import jakarta.persistence.JoinColumn;
 14: import jakarta.persistence.ManyToOne;
 15: import jakarta.persistence.Table;
 16: import lombok.AllArgsConstructor;
 17: import lombok.Data;
 18: import lombok.NoArgsConstructor;
 19:
 20: @Entity
 21: @Table(name = "feedback")
 22: @Data
 23: @NoArgsConstructor
 24: @AllArgsConstructor
 25: public class Feedback {
 26:
 27:     @Id
 28:     @GeneratedValue(strategy = GenerationType.IDENTITY)
 29:     private Long id;
 30:
 31:     @ManyToOne(fetch = FetchType.LAZY)
 32:     @JoinColumn(name = "user_id", nullable = false)
 33:     private User user;
 34:
 35:     @Column(name = "feedback_text", nullable = false, columnDefinition = "TEXT")
 36:     private String feedbackText;
 37:
 38:     @CreationTimestamp
 39:     @Column(name = "submitted_at", nullable = false, updatable = false)
 40:     private LocalDateTime submittedAt;
 41: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\model\Item.java

```
  1: package com.lostandfound.model;
  2:
  3: import java.time.LocalDateTime;
  4:
  5: import org.hibernate.annotations.CreationTimestamp;
  6:
  7: import jakarta.persistence.Column;
  8: import jakarta.persistence.Entity;
  9: import jakarta.persistence.EnumType;
 10: import jakarta.persistence.Enumerated;
 11: import jakarta.persistence.FetchType;
 12: import jakarta.persistence.GeneratedValue;
 13: import jakarta.persistence.GenerationType;
 14: import jakarta.persistence.Id;
 15: import jakarta.persistence.JoinColumn;
 16: import jakarta.persistence.ManyToOne;
 17: import jakarta.persistence.Table;
 18: import jakarta.persistence.Version;
 19: import lombok.AllArgsConstructor;
 20: import lombok.Data;
 21: import lombok.NoArgsConstructor;
 22:
 23: @Entity
 24: @Table(name = "items")
 25: @Data
 26: @NoArgsConstructor
 27: @AllArgsConstructor
 28: public class Item {
 29:
 30:     @Id
 31:     @GeneratedValue(strategy = GenerationType.IDENTITY)
 32:     private Long id;
 33:
 34:     @Column(nullable = false)
 35:     private String name;
 36:
 37:     @Column(nullable = false, columnDefinition = "TEXT")
```

```
    private String description;

    @Column(nullable = false)
    private String location;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    private Status status = Status.LOST;

    @Column(name = "image")
    private String image;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "created_by", nullable = false)
    private User createdBy;

    @CreationTimestamp
    @Column(name = "created_at", nullable = false, updatable = false)
    private LocalDateTime createdAt;

    @Version
    @Column(name = "version")
    private Long version;

    public enum Status {
        LOST, FOUND, CLAIMED
    }
}
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\model\Message.java

```
package com.lostandfound.model;

import java.time.LocalDateTime;

import org.hibernate.annotations.CreationTimestamp;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.FetchType;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.Table;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Table(name = "messages")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Message {

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

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "sender_id", nullable = false)
    private User sender;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "receiver_id", nullable = false)
    private User receiver;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "item_id", nullable = false)
    private Item item;

    @Column(nullable = false, columnDefinition = "TEXT")
    private String message;

    @CreationTimestamp
    @Column(name = "sent_at", nullable = false, updatable = false)
    private LocalDateTime sentAt;
}
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\model\RefreshToken.java

```
  1: package com.lostandfound.model;
  2:
  3: import jakarta.persistence.*;
  4: import lombok.AllArgsConstructor;
  5: import lombok.Builder;
  6: import lombok.Data;
  7: import lombok.NoArgsConstructor;
  8: import org.hibernate.annotations.CreationTimestamp;
  9:
 10: import java.time.Instant;
 11: import java.time.LocalDateTime;
 12:
 13: @Entity
 14: @Table(name = "refresh_tokens")
 15: @Data
 16: @Builder
 17: @NoArgsConstructor
 18: @AllArgsConstructor
 19: public class RefreshToken {
 20:
 21:     @Id
 22:     @GeneratedValue(strategy = GenerationType.IDENTITY)
 23:     private Long id;
 24:
 25:     @OneToOne
 26:     @JoinColumn(name = "user_id", referencedColumnName = "id")
 27:     private User user;
 28:
 29:     @Column(nullable = false, unique = true, length = 500)
 30:     private String token;
 31:
 32:     @Column(nullable = false)
 33:     private Instant expiryDate;
 34:
 35:     @Column(name = "revoked")
 36:     @Builder.Default
 37:     private boolean revoked = false;
 38:
 39:     @CreationTimestamp
 40:     @Column(name = "created_at", nullable = false, updatable = false)
 41:     private LocalDateTime createdAt;
 42:
 43:     @Column(name = "ip_address")
 44:     private String ipAddress;
 45:
 46:     @Column(name = "user_agent")
 47:     private String userAgent;
 48: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\model\User.java

```
  1: package com.lostandfound.model;
  2:
  3: import java.time.LocalDateTime;
  4:
  5: import org.hibernate.annotations.CreationTimestamp;
  6:
  7: import jakarta.persistence.Column;
  8: import jakarta.persistence.Entity;
  9: import jakarta.persistence.EnumType;
 10: import jakarta.persistence.Enumerated;
 11: import jakarta.persistence.GeneratedValue;
 12: import jakarta.persistence.GenerationType;
 13: import jakarta.persistence.Id;
 14: import jakarta.persistence.Table;
 15: import lombok.AllArgsConstructor;
 16: import lombok.Data;
 17: import lombok.NoArgsConstructor;
 18:
 19: @Entity
 20: @Table(name = "users")
 21: @Data
 22: @NoArgsConstructor
 23: @AllArgsConstructor
 24: public class User {
 25:
 26:     @Id
 27:     @GeneratedValue(strategy = GenerationType.IDENTITY)
 28:     private Long id;
 29:
 30:     @Column(nullable = false)
```

```
 31:     private String name;
 32:
 33:     @Column(nullable = false, unique = true)
 34:     private String email;
 35:
 36:     @Column(nullable = false)
 37:     private String password;
 38:
 39:     @Enumerated(EnumType.STRING)
 40:     @Column(nullable = false)
 41:     private Role role = Role.USER;
 42:
 43:     @CreationTimestamp
 44:     @Column(name = "created_at", nullable = false, updatable = false)
 45:     private LocalDateTime createdAt;
 46:
 47:     public enum Role {
 48:         USER, ADMIN
 49:     }
 50: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\repository\ClaimRepository.java

```
  1: package com.lostandfound.repository;
  2:
  3: import java.time.LocalDateTime;
  4: import java.util.List;
  5: import java.util.Optional;
  6:
  7: import org.springframework.data.jpa.repository.JpaRepository;
  8: import org.springframework.data.jpa.repository.Query;
  9: import org.springframework.data.repository.query.Param;
 10: import org.springframework.stereotype.Repository;
 11:
 12: import com.lostandfound.model.Claim;
 13: import com.lostandfound.model.Item;
 14: import com.lostandfound.model.User;
 15:
 16: @Repository
 17: public interface ClaimRepository extends JpaRepository<Claim, Long> {
 18:
 19:     List<Claim> findByClaimedByOrderByClaimedAtDesc(User user);
 20:
 21:     Optional<Claim> findByItemAndClaimedBy(Item item, User user);
 22:
 23:     boolean existsByItemAndClaimedBy(Item item, User user);
 24:
 25:     @Query("SELECT c FROM Claim c WHERE c.claimedAt < :cutoffDate")
 26:     List<Claim> findOldClaims(@Param("cutoffDate") LocalDateTime cutoffDate);
 27:
 28:     List<Claim> findByItem(Item item);
 29:
 30:     List<Claim> findAllByOrderByClaimedAtDesc();
 31: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\repository\FeedbackRepository.java

```
  1: package com.lostandfound.repository;
  2:
  3: import java.util.List;
  4:
  5: import org.springframework.data.jpa.repository.JpaRepository;
  6: import org.springframework.stereotype.Repository;
  7:
  8: import com.lostandfound.model.Feedback;
  9:
 10: @Repository
 11: public interface FeedbackRepository extends JpaRepository<Feedback, Long> {
 12:     List<Feedback> findAllByOrderBySubmittedAtDesc();
 13: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\repository\ItemRepository.java

```
  1: package com.lostandfound.repository;
  2:
  3: import com.lostandfound.model.Item;
  4: import com.lostandfound.model.Item.Status;
  5: import com.lostandfound.model.User;
  6: import org.springframework.data.jpa.repository.JpaRepository;
```

```
  7: import org.springframework.data.jpa.repository.Query;
  8: import org.springframework.data.repository.query.Param;
  9: import org.springframework.stereotype.Repository;
 10:
 11: import java.util.List;
 12:
 13: @Repository
 14: public interface ItemRepository extends JpaRepository<Item, Long> {
 15:
 16:     List<Item> findByCreatedByOrderByCreatedAtDesc(User user);
 17:
 18:     @Query("SELECT i FROM Item i WHERE " +
 19:            "(:search IS NULL OR :search = '' OR " +
 20:            "LOWER(i.name) LIKE LOWER(CONCAT('%', :search, '%')) OR " +
 21:            "LOWER(i.description) LIKE LOWER(CONCAT('%', :search, '%')) OR " +
 22:            "LOWER(i.location) LIKE LOWER(CONCAT('%', :search, '%'))) AND " +
 23:            "(:status IS NULL OR i.status = :status) " +
 24:            "ORDER BY i.createdAt DESC")
 25:     List<Item> searchItems(@Param("search") String search,
 26:                            @Param("status") Status status);
 27:
 28:     List<Item> findAllByOrderByCreatedAtDesc();
 29: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\repository\MessageRepository.java

```
  1: package com.lostandfound.repository;
  2:
  3: import java.util.List;
  4:
  5: import org.springframework.data.jpa.repository.JpaRepository;
  6: import org.springframework.stereotype.Repository;
  7:
  8: import com.lostandfound.model.Message;
  9: import com.lostandfound.model.User;
 10:
 11: @Repository
 12: public interface MessageRepository extends JpaRepository<Message, Long> {
 13:     List<Message> findByReceiverOrderBySentAtDesc(User receiver);
 14:     List<Message> findBySenderOrderBySentAtDesc(User sender);
 15:     List<Message> findAllByOrderBySentAtDesc();
 16: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\repository\RefreshTokenRepository.java

```
  1: package com.lostandfound.repository;
  2:
  3: import com.lostandfound.model.RefreshToken;
  4: import com.lostandfound.model.User;
  5: import org.springframework.data.jpa.repository.JpaRepository;
  6: import org.springframework.data.jpa.repository.Modifying;
  7: import org.springframework.data.jpa.repository.Query;
  8: import org.springframework.stereotype.Repository;
  9:
 10: import java.time.Instant;
 11: import java.util.List;
 12: import java.util.Optional;
 13:
 14: @Repository
 15: public interface RefreshTokenRepository extends JpaRepository<RefreshToken, Long> {
 16:
 17:     Optional<RefreshToken> findByToken(String token);
 18:
 19:     Optional<RefreshToken> findByUser(User user);
 20:
 21:     @Modifying
 22:     @Query("DELETE FROM RefreshToken rt WHERE rt.user = ?1")
 23:     int deleteByUser(User user);
 24:
 25:     @Modifying
 26:     @Query("DELETE FROM RefreshToken rt WHERE rt.expiryDate < ?1")
 27:     int deleteAllExpiredTokens(Instant now);
 28:
 29:     @Query("SELECT rt FROM RefreshToken rt WHERE rt.user = ?1 AND rt.revoked = false")
 30:     List<RefreshToken> findActiveTokensByUser(User user);
 31:
 32:     boolean existsByToken(String token);
 33: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\repository\UserRepository.java

```
  1: package com.lostandfound.repository;
  2:
  3: import com.lostandfound.model.User;
  4: import org.springframework.data.jpa.repository.JpaRepository;
  5: import org.springframework.stereotype.Repository;
  6:
  7: import java.util.Optional;
  8: import java.util.List;
  9:
 10: @Repository
 11: public interface UserRepository extends JpaRepository<User, Long> {
 12:     Optional<User> findByEmail(String email);
 13:     Boolean existsByEmail(String email);
 14:     List<User> findByRoleNot(User.Role role);
 15: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\scheduler\ClaimCleanupScheduler.java

```
  1: package com.lostandfound.scheduler;
  2:
  3: import com.lostandfound.model.Claim;
  4: import com.lostandfound.model.Item;
  5: import com.lostandfound.repository.ClaimRepository;
  6: import com.lostandfound.repository.ItemRepository;
  7: import lombok.RequiredArgsConstructor;
  8: import org.slf4j.Logger;
  9: import org.slf4j.LoggerFactory;
 10: import org.springframework.scheduling.annotation.Scheduled;
 11: import org.springframework.stereotype.Component;
 12: import org.springframework.transaction.annotation.Transactional;
 13:
 14: import java.time.LocalDateTime;
 15: import java.util.List;
 16:
 17: @Component
 18: @RequiredArgsConstructor
 19: public class ClaimCleanupScheduler {
 20:
 21:     private static final Logger logger = LoggerFactory.getLogger(ClaimCleanupScheduler.cla
    | ss);
 22:
 23:     private final ClaimRepository claimRepository;
 24:     private final ItemRepository itemRepository;
 25:
 26:     @Scheduled(cron = "0 0 2 * * ?") // Run every day at 2 AM
 27:     @Transactional
 28:     public void cleanupOldClaims() {
 29:         logger.info("Starting cleanup of old claims...");
 30:
 31:         try {
 32:             LocalDateTime cutoffDate = LocalDateTime.now().minusDays(7);
 33:             List<Claim> oldClaims = claimRepository.findOldClaims(cutoffDate);
 34:
 35:             int claimsDeleted = oldClaims.size();
 36:
 37:             for (Claim claim : oldClaims) {
 38:                 Item item = claim.getItem();
 39:                 claimRepository.delete(claim);
 40:
 41:                 // Update item status if it was claimed
 42:                 if (item.getStatus() == Item.Status.CLAIMED) {
 43:                     item.setStatus(Item.Status.FOUND);
 44:                     itemRepository.save(item);
 45:                 }
 46:             }
 47:
 48:             logger.info("Deleted {} old claims and updated item statuses", claimsDeleted);
 49:         } catch (Exception e) {
 50:             logger.error("Error during claim cleanup", e);
 51:         }
 52:     }
 53: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\scheduler\TokenCleanupScheduler.java

```
  1: package com.lostandfound.scheduler;
  2:
  3: import com.lostandfound.service.RefreshTokenService;
  4: import lombok.RequiredArgsConstructor;
```

```
  5: import org.slf4j.Logger;
  6: import org.slf4j.LoggerFactory;
  7: import org.springframework.scheduling.annotation.Scheduled;
  8: import org.springframework.stereotype.Component;
  9: import org.springframework.transaction.annotation.Transactional;
 10:
 11: @Component
 12: @RequiredArgsConstructor
 13: public class TokenCleanupScheduler {
 14:
 15:     private static final Logger logger = LoggerFactory.getLogger(TokenCleanupScheduler.cla
    | ss);
 16:
 17:     private final RefreshTokenService refreshTokenService;
 18:
 19:     // Run every day at 3 AM
 20:     @Scheduled(cron = "0 0 3 * * ?")
 21:     @Transactional
 22:     public void cleanupExpiredTokens() {
 23:         logger.info("Starting cleanup of expired refresh tokens...");
 24:
 25:         try {
 26:             refreshTokenService.deleteExpiredTokens();
 27:             logger.info("Successfully cleaned up expired refresh tokens");
 28:         } catch (Exception e) {
 29:             logger.error("Error during token cleanup", e);
 30:         }
 31:     }
 32: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\security\CustomUserDetailsService.java

```
  1: package com.lostandfound.security;
  2:
  3: import com.lostandfound.exception.ResourceNotFoundException;
  4: import com.lostandfound.model.User;
  5: import com.lostandfound.repository.UserRepository;
  6: import lombok.RequiredArgsConstructor;
  7: import org.springframework.security.core.userdetails.UserDetails;
  8: import org.springframework.security.core.userdetails.UserDetailsService;
  9: import org.springframework.security.core.userdetails.UsernameNotFoundException;
 10: import org.springframework.stereotype.Service;
 11: import org.springframework.transaction.annotation.Transactional;
 12:
 13: @Service
 14: @RequiredArgsConstructor
 15: public class CustomUserDetailsService implements UserDetailsService {
 16:
 17:     private final UserRepository userRepository;
 18:
 19:     @Override
 20:     @Transactional
 21:     public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
 22:         User user = userRepository.findByEmail(email)
 23:                 .orElseThrow(() -> new UsernameNotFoundException("User not found with emai
    | l: " + email));
 24:
 25:         return UserPrincipal.create(user);
 26:     }
 27:
 28:     @Transactional
 29:     public UserDetails loadUserById(Long id) {
 30:         User user = userRepository.findById(id)
 31:                 .orElseThrow(() -> new ResourceNotFoundException("User", "id", id));
 32:
 33:         return UserPrincipal.create(user);
 34:     }
 35: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\security\JwtAuthenticationFilter.java

```
  1: package com.lostandfound.security;
  2:
  3: import com.lostandfound.util.CookieUtil;
  4: import jakarta.servlet.FilterChain;
  5: import jakarta.servlet.ServletException;
  6: import jakarta.servlet.http.HttpServletRequest;
  7: import jakarta.servlet.http.HttpServletResponse;
  8: import lombok.RequiredArgsConstructor;
  9: import org.slf4j.Logger;
 10: import org.slf4j.LoggerFactory;
```

```
 11: import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
 12: import org.springframework.security.core.context.SecurityContextHolder;
 13: import org.springframework.security.core.userdetails.UserDetails;
 14: import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
 15: import org.springframework.stereotype.Component;
 16: import org.springframework.util.StringUtils;
 17: import org.springframework.web.filter.OncePerRequestFilter;
 18:
 19: import java.io.IOException;
 20:
 21: @Component
 22: @RequiredArgsConstructor
 23: public class JwtAuthenticationFilter extends OncePerRequestFilter {
 24:
 25:     private static final Logger logger = LoggerFactory.getLogger(JwtAuthenticationFilter.c
    | lass);
 26:
 27:     private final JwtTokenProvider tokenProvider;
 28:     private final CustomUserDetailsService customUserDetailsService;
 29:     private final CookieUtil cookieUtil;
 30:
 31:     @Override
 32:     protected void doFilterInternal(HttpServletRequest request,
 33:                                     HttpServletResponse response,
 34:                                     FilterChain filterChain) throws ServletException, IOEx
    | ception {
 35:         try {
 36:             // Try to get JWT from cookie first, then fall back to header
 37:             String jwt = getJwtFromCookie(request);
 38:             if (!StringUtils.hasText(jwt)) {
 39:                 jwt = getJwtFromHeader(request);
 40:             }
 41:
 42:             if (StringUtils.hasText(jwt) && tokenProvider.validateToken(jwt)) {
 43:                 Long userId = tokenProvider.getUserIdFromToken(jwt);
 44:                 UserDetails userDetails = customUserDetailsService.loadUserById(userId);
 45:
 46:                 UsernamePasswordAuthenticationToken authentication =
 47:                         new UsernamePasswordAuthenticationToken(userDetails, null, userDet
    | ails.getAuthorities());
 48:                 authentication.setDetails(new WebAuthenticationDetailsSource().buildDetail
    | s(request));
 49:
 50:                 SecurityContextHolder.getContext().setAuthentication(authentication);
 51:                 logger.debug("Set authentication for user: {}", userId);
 52:             }
 53:         } catch (Exception ex) {
 54:             logger.error("Could not set user authentication in security context", ex);
 55:         }
 56:
 57:         filterChain.doFilter(request, response);
 58:     }
 59:
 60:     /**
 61:      * Get JWT from cookie
 62:      */
 63:     private String getJwtFromCookie(HttpServletRequest request) {
 64:         return cookieUtil.getAccessToken(request).orElse(null);
 65:     }
 66:
 67:     /**
 68:      * Get JWT from Authorization header (fallback for API clients)
 69:      */
 70:     private String getJwtFromHeader(HttpServletRequest request) {
 71:         String bearerToken = request.getHeader("Authorization");
 72:         if (StringUtils.hasText(bearerToken) && bearerToken.startsWith("Bearer ")) {
 73:             return bearerToken.substring(7);
 74:         }
 75:         return null;
 76:     }
 77: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\security\JwtTokenProvider.java

```
  1: package com.lostandfound.security;
  2:
  3: import io.jsonwebtoken.*;
  4: import io.jsonwebtoken.security.Keys;
  5: import org.slf4j.Logger;
  6: import org.slf4j.LoggerFactory;
  7: import org.springframework.beans.factory.annotation.Value;
  8: import org.springframework.security.core.Authentication;
  9: import org.springframework.stereotype.Component;
```

```
 10:
 11: import javax.crypto.SecretKey;
 12: import java.util.Date;
 13:
 14: @Component
 15: public class JwtTokenProvider {
 16:
 17:     private static final Logger logger = LoggerFactory.getLogger(JwtTokenProvider.class);
 18:
 19:     @Value("${jwt.secret}")
 20:     private String jwtSecret;
 21:
 22:     @Value("${jwt.expiration}")
 23:     private long jwtExpirationMs;
 24:
 25:     private SecretKey getSigningKey() {
 26:         return Keys.hmacShaKeyFor(jwtSecret.getBytes());
 27:     }
 28:
 29:     public String generateToken(Authentication authentication) {
 30:         UserPrincipal userPrincipal = (UserPrincipal) authentication.getPrincipal();
 31:
 32:         Date now = new Date();
 33:         Date expiryDate = new Date(now.getTime() + jwtExpirationMs);
 34:
 35:         return Jwts.builder()
 36:                 .subject(Long.toString(userPrincipal.getId()))
 37:                 .claim("email", userPrincipal.getEmail())
 38:                 .claim("role", userPrincipal.getAuthorities().iterator().next().getAuthori
    | ty())
 39:                 .issuedAt(now)
 40:                 .expiration(expiryDate)
 41:                 .signWith(getSigningKey())
 42:                 .compact();
 43:     }
 44:
 45:     public Long getUserIdFromToken(String token) {
 46:         Claims claims = Jwts.parser()
 47:                 .verifyWith(getSigningKey())
 48:                 .build()
 49:                 .parseSignedClaims(token)
 50:                 .getPayload();
 51:
 52:         return Long.parseLong(claims.getSubject());
 53:     }
 54:
 55:     public boolean validateToken(String token) {
 56:         try {
 57:             Jwts.parser()
 58:                 .verifyWith(getSigningKey())
 59:                 .build()
 60:                 .parseSignedClaims(token);
 61:             return true;
 62:         } catch (JwtException | IllegalArgumentException e) {
 63:             logger.error("Invalid JWT token: {}", e.getMessage());
 64:             return false;
 65:         }
 66:     }
 67: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\security\RateLimitFilter.java

```
  1: package com.lostandfound.security;
  2:
  3: import com.fasterxml.jackson.databind.ObjectMapper;
  4: import com.lostandfound.config.RateLimitConfig;
  5: import com.lostandfound.dto.response.ApiResponse;
  6: import io.github.bucket4j.Bucket;
  7: import io.github.bucket4j.ConsumptionProbe;
  8: import jakarta.servlet.FilterChain;
  9: import jakarta.servlet.ServletException;
 10: import jakarta.servlet.http.HttpServletRequest;
 11: import jakarta.servlet.http.HttpServletResponse;
 12: import lombok.RequiredArgsConstructor;
 13: import org.slf4j.Logger;
 14: import org.slf4j.LoggerFactory;
 15: import org.springframework.beans.factory.annotation.Value;
 16: import org.springframework.http.HttpStatus;
 17: import org.springframework.http.MediaType;
 18: import org.springframework.stereotype.Component;
 19: import org.springframework.web.filter.OncePerRequestFilter;
 20:
 21: import java.io.IOException;
```

```
22:
23: @Component
24: @RequiredArgsConstructor
25: public class RateLimitFilter extends OncePerRequestFilter {
26:
27:     private static final Logger logger = LoggerFactory.getLogger(RateLimitFilter.class);
28:
29:     private final RateLimitConfig rateLimitConfig;
30:     private final ObjectMapper objectMapper;
31:
32:     @Value("${rate.limit.enabled:true}")
33:     private boolean rateLimitEnabled;
34:
35:     @Override
36:     protected void doFilterInternal(HttpServletRequest request,
37:                                     HttpServletResponse response,
38:                                     FilterChain filterChain) throws ServletException, IOEx
   | ception {
39:
40:         // Skip rate limiting if disabled
41:         if (!rateLimitEnabled) {
42:             filterChain.doFilter(request, response);
43:             return;
44:         }
45:
46:         String path = request.getRequestURI();
47:         String clientIp = getClientIP(request);
48:
49:         // Determine rate limit type based on path
50:         RateLimitConfig.RateLimitType limitType = determineRateLimitType(path);
51:
52:         // Create unique key: IP + Path pattern
53:         String bucketKey = clientIp + ":" + limitType.name();
54:
55:         // Get or create bucket for this key
56:         Bucket bucket = rateLimitConfig.resolveBucket(bucketKey, limitType);
57:
58:         // Try to consume a token
59:         ConsumptionProbe probe = bucket.tryConsumeAndReturnRemaining(1);
60:
61:         if (probe.isConsumed()) {
62:             // Request allowed - add rate limit headers
63:             response.addHeader("X-Rate-Limit-Remaining", String.valueOf(probe.getRemaining
   | Tokens()));
64:             response.addHeader("X-Rate-Limit-Retry-After-Seconds",
65:                     String.valueOf(limitType.getRefillDuration().getSeconds()));
66:
67:             filterChain.doFilter(request, response);
68:         } else {
69:             // Rate limit exceeded
70:             long waitForRefill = probe.getNanosToWaitForRefill() / 1_000_000_000;
71:
72:             logger.warn("Rate limit exceeded for IP: {} on path: {}", clientIp, path);
73:
74:             response.setStatus(HttpStatus.TOO_MANY_REQUESTS.value());
75:             response.setContentType(MediaType.APPLICATION_JSON_VALUE);
76:             response.addHeader("X-Rate-Limit-Retry-After-Seconds", String.valueOf(waitForR
   | efill));
77:
78:             ApiResponse apiResponse = ApiResponse.builder()
79:                     .success(false)
80:                     .message("Rate limit exceeded. Please try again in " + waitForRefill +
   |  " seconds.")
81:                     .build();
82:
83:             response.getWriter().write(objectMapper.writeValueAsString(apiResponse));
84:         }
85:     }
86:
87:     /**
88:      * Determine rate limit type based on request path
89:      */
90:     private RateLimitConfig.RateLimitType determineRateLimitType(String path) {
91:         if (path.startsWith("/api/auth/")) {
92:             return RateLimitConfig.RateLimitType.AUTH;
93:         } else if (path.startsWith("/admin/")) {
94:             return RateLimitConfig.RateLimitType.ADMIN;
95:         } else if (path.startsWith("/items") && path.contains("/image")) {
96:             return RateLimitConfig.RateLimitType.UPLOAD;
97:         } else if (path.startsWith("/uploads/") || path.startsWith("/static/")) {
98:             return RateLimitConfig.RateLimitType.PUBLIC;
99:         } else {
100:             return RateLimitConfig.RateLimitType.API;
101:         }
102:     }
```

```
103:
104:     /**
105:      * Extract client IP address with proper proxy handling
106:      */
107:     private String getClientIP(HttpServletRequest request) {
108:         // Only trust X-Forwarded-For if behind a known proxy
109:         String xfHeader = request.getHeader("X-Forwarded-For");
110:
111:         // Check if request is from trusted proxy
112:         String remoteAddr = request.getRemoteAddr();
113:         boolean isTrustedProxy = isTrustedProxy(remoteAddr);
114:
115:         if (isTrustedProxy && xfHeader != null && !xfHeader.isEmpty()) {
116:             // X-Forwarded-For can contain multiple IPs, get the first one
117:             String clientIp = xfHeader.split(",")[0].trim();
118:
119:             // Validate IP format to prevent injection
120:             if (isValidIP(clientIp)) {
121:                 return clientIp;
122:             }
123:         }
124:
125:         return remoteAddr;
126:     }
127:
128:     /**
129:      * Check if IP is from trusted proxy
130:      */
131:     private boolean isTrustedProxy(String ip) {
132:         // Add your trusted proxy IPs here (e.g., load balancer, reverse proxy)
133:         // For local development, trust localhost
134:         return "127.0.0.1".equals(ip) ||
135:                 "0:0:0:0:0:0:0:1".equals(ip) ||
136:                 "::1".equals(ip);
137:         // In production, add your actual proxy IPs:
138:         // return Arrays.asList("10.0.0.1", "172.16.0.1").contains(ip);
139:     }
140:
141:     /**
142:      * Validate IP address format
143:      */
144:     private boolean isValidIP(String ip) {
145:         if (ip == null || ip.isEmpty()) {
146:             return false;
147:         }
148:
149:         // Simple validation for IPv4
150:         String ipv4Pattern = "^((25[0-5]|(2[0-4]|1\\d|[1-9]|)\\d)\\.?\\b){4}$";
151:
152:         // Simple validation for IPv6
153:         String ipv6Pattern = "^(([0-9a-fA-F]{1,4}:){7}[0-9a-fA-F]{1,4}|" +
154:                 "([0-9a-fA-F]{1,4}:){1,7}:|" +
155:                 "([0-9a-fA-F]{1,4}:){1,6}:[0-9a-fA-F]{1,4}|" +
156:                 "([0-9a-fA-F]{1,4}:){1,5}(:[0-9a-fA-F]{1,4}){1,2})$";
157:
158:         return ip.matches(ipv4Pattern) || ip.matches(ipv6Pattern);
159:     }
160:
161:     /**
162:      * Skip rate limiting for health check endpoints
163:      */
164:     @Override
165:     protected boolean shouldNotFilter(HttpServletRequest request) {
166:         String path = request.getRequestURI();
167:         return path.startsWith("/actuator/health") ||
168:                 path.startsWith("/actuator/info");
169:     }
170: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\security\UserPrincipal.java

```
  1: package com.lostandfound.security;
  2:
  3: import com.lostandfound.model.User;
  4: import lombok.AllArgsConstructor;
  5: import lombok.Data;
  6: import org.springframework.security.core.GrantedAuthority;
  7: import org.springframework.security.core.authority.SimpleGrantedAuthority;
  8: import org.springframework.security.core.userdetails.UserDetails;
  9:
 10: import java.util.Collection;
 11: import java.util.Collections;
 12:
```

```
13: @Data
14: @AllArgsConstructor
15: public class UserPrincipal implements UserDetails {
16:
17:     private Long id;
18:     private String name;
19:     private String email;
20:     private String password;
21:     private Collection<? extends GrantedAuthority> authorities;
22:
23:     public static UserPrincipal create(User user) {
24:         Collection<GrantedAuthority> authorities = Collections.singleton(
25:             new SimpleGrantedAuthority("ROLE_" + user.getRole().name())
26:         );
27:
28:         return new UserPrincipal(
29:             user.getId(),
30:             user.getName(),
31:             user.getEmail(),
32:             user.getPassword(),
33:             authorities
34:         );
35:     }
36:
37:     @Override
38:     public String getUsername() {
39:         return email;
40:     }
41:
42:     @Override
43:     public boolean isAccountNonExpired() {
44:         return true;
45:     }
46:
47:     @Override
48:     public boolean isAccountNonLocked() {
49:         return true;
50:     }
51:
52:     @Override
53:     public boolean isCredentialsNonExpired() {
54:         return true;
55:     }
56:
57:     @Override
58:     public boolean isEnabled() {
59:         return true;
60:     }
61: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\service\ClaimService.java

```
  1: package com.lostandfound.service;
  2:
  3: import com.lostandfound.dto.response.ApiResponse;
  4: import com.lostandfound.dto.response.ClaimResponse;
  5: import com.lostandfound.exception.BadRequestException;
  6: import com.lostandfound.exception.ResourceNotFoundException;
  7: import com.lostandfound.exception.UnauthorizedException;
  8: import com.lostandfound.model.Claim;
  9: import com.lostandfound.model.Item;
 10: import com.lostandfound.model.User;
 11: import com.lostandfound.repository.ClaimRepository;
 12: import com.lostandfound.repository.ItemRepository;
 13: import com.lostandfound.repository.UserRepository;
 14: import com.lostandfound.security.UserPrincipal;
 15: import lombok.RequiredArgsConstructor;
 16: import org.slf4j.Logger;
 17: import org.slf4j.LoggerFactory;
 18: import org.springframework.stereotype.Service;
 19: import org.springframework.transaction.annotation.Transactional;
 20:
 21: import java.util.List;
 22: import java.util.stream.Collectors;
 23:
 24: @Service
 25: @RequiredArgsConstructor
 26: public class ClaimService {
 27:
 28:     private static final Logger logger = LoggerFactory.getLogger(ClaimService.class);
 29:
 30:     private final ClaimRepository claimRepository;
 31:     private final ItemRepository itemRepository;
```

```
    private final UserRepository userRepository;

    @Transactional
    public ApiResponse claimItem(Long itemId, UserPrincipal currentUser) {
        // Validate user is authenticated
        if (currentUser == null) {
            throw new UnauthorizedException("You must be logged in to claim an item");
        }

        User user = userRepository.findById(currentUser.getId())
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
   | .getId()));

        // Use pessimistic locking to prevent race conditions
        Item item = itemRepository.findById(itemId)
                .orElseThrow(() -> new ResourceNotFoundException("Item", "id", itemId));

        // Synchronize on item to prevent concurrent claims
        synchronized (this) {
            // Re-fetch with lock to ensure we have latest state
            item = itemRepository.findById(itemId)
                    .orElseThrow(() -> new ResourceNotFoundException("Item", "id", itemId)
   | );

            // Check if item is already claimed
            if (item.getStatus() == Item.Status.CLAIMED) {
                throw new BadRequestException("This item has already been claimed by someo
   | ne else");
            }

            // Check if user is trying to claim their own item
            if (item.getCreatedBy().getId().equals(currentUser.getId())) {
                throw new BadRequestException("You cannot claim your own item");
            }

            // Check if user has already claimed this item
            if (claimRepository.existsByItemAndClaimedBy(item, user)) {
                throw new BadRequestException("You have already submitted a claim for this
   |  item");
            }

            // Create new claim
            Claim claim = new Claim();
            claim.setItem(item);
            claim.setClaimedBy(user);
            claim.setClaimantName(user.getName());
            claim.setClaimantEmail(user.getEmail());

            claimRepository.save(claim);

            // Update item status
            item.setStatus(Item.Status.CLAIMED);
            itemRepository.save(item);

            logger.info("User ID {} claimed item {}", user.getId(), itemId);
        }

        return ApiResponse.builder()
                .success(true)
                .message("Item claimed successfully. The item owner will be notified.")
                .build();
    }

    @Transactional(readOnly = true)
    public List<ClaimResponse> getUserClaims(UserPrincipal currentUser) {
        if (currentUser == null) {
            // For admin use - return all claims
            List<Claim> claims = claimRepository.findAllByOrderByClaimedAtDesc();
            return claims.stream()
                    .map(this::mapToClaimResponse)
                    .collect(Collectors.toList());
        }

        User user = userRepository.findById(currentUser.getId())
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
   | .getId()));

        List<Claim> claims = claimRepository.findByClaimedByOrderByClaimedAtDesc(user);
        return claims.stream()
                .map(this::mapToClaimResponse)
                .collect(Collectors.toList());
    }

    @Transactional
    public void deleteClaim(Long claimId) {
```

```
112:         Claim claim = claimRepository.findById(claimId)
113:                 .orElseThrow(() -> new ResourceNotFoundException("Claim", "id", claimId));
114:
115:         Item item = claim.getItem();
116:
117:         claimRepository.delete(claim);
118:
119:         // Update item status back to FOUND if it was CLAIMED and no other claims exist
120:         if (item.getStatus() == Item.Status.CLAIMED) {
121:             List<Claim> remainingClaims = claimRepository.findByItem(item);
122:             if (remainingClaims.isEmpty()) {
123:                 item.setStatus(Item.Status.FOUND);
124:                 itemRepository.save(item);
125:             }
126:         }
127:
128:         logger.info("Deleted claim {} for item {}", claimId, item.getId());
129:     }
130:
131:     @Transactional
132:     public void deleteClaimByUser(Long claimId, UserPrincipal currentUser) {
133:         if (currentUser == null) {
134:             throw new UnauthorizedException("You must be logged in to delete a claim");
135:         }
136:
137:         Claim claim = claimRepository.findById(claimId)
138:                 .orElseThrow(() -> new ResourceNotFoundException("Claim", "id", claimId));
139:
140:         // Check if user owns this claim or is admin
141:         User user = userRepository.findById(currentUser.getId())
142:                 .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
    | .getId()));
143:
144:         boolean isAdmin = user.getRole() == User.Role.ADMIN;
145:         boolean isOwner = claim.getClaimedBy().getId().equals(currentUser.getId());
146:
147:         if (!isAdmin && !isOwner) {
148:             throw new UnauthorizedException("You don't have permission to delete this clai
    | m");
149:         }
150:
151:         deleteClaim(claimId);
152:     }
153:
154:     private ClaimResponse mapToClaimResponse(Claim claim) {
155:         return ClaimResponse.builder()
156:                 .id(claim.getId())
157:                 .itemId(claim.getItem().getId())
158:                 .itemName(claim.getItem().getName())
159:                 .description(claim.getItem().getDescription())
160:                 .location(claim.getItem().getLocation())
161:                 .claimedBy(claim.getClaimedBy().getId())
162:                 .claimerName(claim.getClaimedBy().getName())
163:                 .claimantName(claim.getClaimantName())
164:                 .claimantEmail(claim.getClaimantEmail())
165:                 .claimedAt(claim.getClaimedAt())
166:                 .build();
167:     }
168: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\service\FeedbackService.java

```
  1: package com.lostandfound.service;
  2:
  3: import com.lostandfound.dto.request.FeedbackRequest;
  4: import com.lostandfound.dto.response.ApiResponse;
  5: import com.lostandfound.dto.response.FeedbackResponse;
  6: import com.lostandfound.exception.ResourceNotFoundException;
  7: import com.lostandfound.model.Feedback;
  8: import com.lostandfound.model.User;
  9: import com.lostandfound.repository.FeedbackRepository;
 10: import com.lostandfound.repository.UserRepository;
 11: import com.lostandfound.security.UserPrincipal;
 12: import lombok.RequiredArgsConstructor;
 13: import org.springframework.stereotype.Service;
 14: import org.springframework.transaction.annotation.Transactional;
 15:
 16: import java.util.List;
 17: import java.util.stream.Collectors;
 18:
 19: @Service
 20: @RequiredArgsConstructor
 21: public class FeedbackService {
```

```
 22:
 23:     private final FeedbackRepository feedbackRepository;
 24:     private final UserRepository userRepository;
 25:
 26:     @Transactional
 27:     public ApiResponse submitFeedback(FeedbackRequest request, UserPrincipal currentUser)
    | {
 28:         User user = userRepository.findById(currentUser.getId())
 29:                 .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
    | .getId()));
 30:
 31:         Feedback feedback = new Feedback();
 32:         feedback.setUser(user);
 33:         feedback.setFeedbackText(request.getFeedback());
 34:
 35:         feedbackRepository.save(feedback);
 36:
 37:         return ApiResponse.builder()
 38:                 .success(true)
 39:                 .message("Thank you for your feedback!")
 40:                 .build();
 41:     }
 42:
 43:     @Transactional(readOnly = true)
 44:     public List<FeedbackResponse> getAllFeedback() {
 45:         List<Feedback> feedbacks = feedbackRepository.findAllByOrderBySubmittedAtDesc();
 46:         return feedbacks.stream()
 47:                 .map(this::mapToFeedbackResponse)
 48:                 .collect(Collectors.toList());
 49:     }
 50:
 51:     @Transactional
 52:     public void deleteFeedback(Long feedbackId) {
 53:         Feedback feedback = feedbackRepository.findById(feedbackId)
 54:                 .orElseThrow(() -> new ResourceNotFoundException("Feedback", "id", feedbac
    | kId));
 55:
 56:         feedbackRepository.delete(feedback);
 57:     }
 58:
 59:     private FeedbackResponse mapToFeedbackResponse(Feedback feedback) {
 60:         return FeedbackResponse.builder()
 61:                 .id(feedback.getId())
 62:                 .userId(feedback.getUser().getId())
 63:                 .userName(feedback.getUser().getName())
 64:                 .feedbackText(feedback.getFeedbackText())
 65:                 .submittedAt(feedback.getSubmittedAt())
 66:                 .build();
 67:     }
 68: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\service\FileStorageService.java

```
  1: package com.lostandfound.service;
  2:
  3: import com.lostandfound.config.FileStorageProperties;
  4: import com.lostandfound.exception.BadRequestException;
  5: import lombok.RequiredArgsConstructor;
  6: import org.slf4j.Logger;
  7: import org.slf4j.LoggerFactory;
  8: import org.springframework.stereotype.Service;
  9: import org.springframework.util.StringUtils;
 10: import org.springframework.web.multipart.MultipartFile;
 11:
 12: import java.io.IOException;
 13: import java.nio.file.Files;
 14: import java.nio.file.Path;
 15: import java.nio.file.Paths;
 16: import java.nio.file.StandardCopyOption;
 17: import java.util.UUID;
 18:
 19: @Service
 20: @RequiredArgsConstructor
 21: public class FileStorageService {
 22:
 23:     private static final Logger logger = LoggerFactory.getLogger(FileStorageService.class)
    | ;
 24:     private final FileStorageProperties fileStorageProperties;
 25:
 26:     public String storeFile(MultipartFile file) {
 27:         // Normalize file name
 28:         String originalFileName = StringUtils.cleanPath(file.getOriginalFilename());
 29:
```

```
30:         try {
31:             // Check if the file's name contains invalid characters
32:             if (originalFileName.contains("..")) {
33:                 throw new BadRequestException("Filename contains invalid path sequence " +
   |  originalFileName);
34:             }
35:
36:             // Create unique filename
37:             String fileExtension = "";
38:             if (originalFileName.contains(".")) {
39:                 fileExtension = originalFileName.substring(originalFileName.lastIndexOf(".
   | "));
40:             }
41:             String uniqueFileName = UUID.randomUUID().toString() + "_" + originalFileName;
42:
43:             // Create directory if it doesn't exist
44:             Path fileStorageLocation = Paths.get(fileStorageProperties.getUploadDir())
45:                     .toAbsolutePath().normalize();
46:             Files.createDirectories(fileStorageLocation);
47:
48:             // Copy file to the target location
49:             Path targetLocation = fileStorageLocation.resolve(uniqueFileName);
50:             Files.copy(file.getInputStream(), targetLocation, StandardCopyOption.REPLACE_E
   | XISTING);
51:
52:             return "uploads/" + uniqueFileName;
53:         } catch (IOException ex) {
54:             throw new BadRequestException("Could not store file " + originalFileName + ".
   | Please try again!");
55:         }
56:     }
57:
58:     public void deleteFile(String filePath) {
59:         try {
60:             if (filePath != null && filePath.startsWith("uploads/")) {
61:                 String fileName = filePath.replace("uploads/", "");
62:
63:                 // Prevent path traversal
64:                 if (fileName.contains("..") || fileName.contains("/") || fileName.contains
   | ("\\")) {
65:                     logger.error("Invalid file path detected: {}", filePath);
66:                     return;
67:                 }
68:
69:                 Path fileStorageLocation = Paths.get(fileStorageProperties.getUploadDir())
70:                         .toAbsolutePath().normalize();
71:                 Path targetLocation = fileStorageLocation.resolve(fileName).normalize();
72:
73:                 // Ensure the resolved path is still within the upload directory
74:                 if (!targetLocation.startsWith(fileStorageLocation)) {
75:                     logger.error("Path traversal attempt detected: {}", filePath);
76:                     return;
77:                 }
78:
79:                 Files.deleteIfExists(targetLocation);
80:             }
81:         } catch (IOException ex) {
82:             // Log error but don't throw exception
83:             logger.error("Could not delete file: {}", filePath, ex);
84:         }
85:     }
86: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\service\ItemService.java

```
  1: package com.lostandfound.service;
  2:
  3: import com.lostandfound.dto.request.ItemRequest;
  4: import com.lostandfound.dto.response.ItemResponse;
  5: import com.lostandfound.exception.BadRequestException;
  6: import com.lostandfound.exception.ResourceNotFoundException;
  7: import com.lostandfound.exception.UnauthorizedException;
  8: import com.lostandfound.model.Item;
  9: import com.lostandfound.model.User;
 10: import com.lostandfound.repository.ClaimRepository;
 11: import com.lostandfound.repository.ItemRepository;
 12: import com.lostandfound.repository.MessageRepository;
 13: import com.lostandfound.repository.UserRepository;
 14: import com.lostandfound.security.UserPrincipal;
 15: import lombok.RequiredArgsConstructor;
 16: import org.slf4j.Logger;
 17: import org.slf4j.LoggerFactory;
 18: import org.springframework.stereotype.Service;
```

```
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.multipart.MultipartFile;

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

@Service
@RequiredArgsConstructor
public class ItemService {

    private static final Logger logger = LoggerFactory.getLogger(ItemService.class);

    private final ItemRepository itemRepository;
    private final UserRepository userRepository;
    private final ClaimRepository claimRepository;
    private final MessageRepository messageRepository;
    private final FileStorageService fileStorageService;

    @Transactional
    public ItemResponse createItem(ItemRequest request, MultipartFile image, UserPrincipal currentUser) {
        if (currentUser == null) {
            throw new UnauthorizedException("You must be logged in to create an item");
        }

        User user = userRepository.findById(currentUser.getId())
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser.getId()));

        // Validate status
        Item.Status status;
        try {
            status = Item.Status.valueOf(request.getStatus().toUpperCase());
        } catch (IllegalArgumentException e) {
            throw new BadRequestException("Invalid status. Must be LOST or FOUND");
        }

        // Only LOST or FOUND allowed for new items
        if (status == Item.Status.CLAIMED) {
            throw new BadRequestException("Cannot create item with CLAIMED status");
        }

        String imagePath = null;
        if (image != null && !image.isEmpty()) {
            imagePath = fileStorageService.storeFile(image);
        }

        Item item = new Item();
        item.setName(request.getName().trim());
        item.setDescription(request.getDescription().trim());
        item.setLocation(request.getLocation().trim());
        item.setStatus(status);
        item.setImage(imagePath);
        item.setCreatedBy(user);

        item = itemRepository.save(item);
        logger.info("User ID {} created new item: {}", user.getId(), item.getId());

        return mapToItemResponse(item);
    }

    @Transactional(readOnly = true)
    public ItemResponse getItemById(Long itemId) {
        Item item = itemRepository.findById(itemId)
                .orElseThrow(() -> new ResourceNotFoundException("Item", "id", itemId));

        return mapToItemResponse(item);
    }

    @Transactional(readOnly = true)
    public List<ItemResponse> searchItems(String search, String status) {
        Item.Status itemStatus = null;
        if (status != null && !status.isEmpty() && !status.equalsIgnoreCase("all")) {
            try {
                itemStatus = Item.Status.valueOf(status.toUpperCase());
            } catch (IllegalArgumentException e) {
                throw new BadRequestException("Invalid status: " + status + ". Must be LOST, FOUND, or CLAIMED");
            }
        }

        // Sanitize search input to prevent wildcard injection
        String sanitizedSearch = sanitizeSearchInput(search);

        List<Item> items = itemRepository.searchItems(
```

```
                sanitizedSearch,
                itemStatus
        );

        return items.stream()
                .map(this::mapToItemResponse)
                .collect(Collectors.toList());
    }

    /**
     * Sanitize search input to prevent SQL wildcard injection
     */
    private String sanitizeSearchInput(String search) {
        if (search == null || search.trim().isEmpty()) {
            return "";
        }

        // Remove or escape SQL wildcards
        String sanitized = search.trim()
                .replace("\\", "\\\\")  // Escape backslash first
                .replace("%", "\\%")     // Escape %
                .replace("_", "\\_");    // Escape _

        // Limit length to prevent DoS
        if (sanitized.length() > 100) {
            sanitized = sanitized.substring(0, 100);
        }

        return sanitized;
    }

    @Transactional(readOnly = true)
    public List<ItemResponse> getUserItems(UserPrincipal currentUser) {
        if (currentUser == null) {
            throw new UnauthorizedException("You must be logged in to view your items");
        }

        User user = userRepository.findById(currentUser.getId())
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
    | .getId()));

        List<Item> items = itemRepository.findByCreatedByOrderByCreatedAtDesc(user);
        return items.stream()
                .map(this::mapToItemResponse)
                .collect(Collectors.toList());
    }

    @Transactional
    public ItemResponse updateItem(Long itemId, ItemRequest request, MultipartFile image,
    | UserPrincipal currentUser) {
        if (currentUser == null) {
            throw new UnauthorizedException("You must be logged in to update an item");
        }

        Item item = itemRepository.findById(itemId)
                .orElseThrow(() -> new ResourceNotFoundException("Item", "id", itemId));

        User user = userRepository.findById(currentUser.getId())
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
    | .getId()));

        // Check permissions
        boolean isAdmin = user.getRole() == User.Role.ADMIN;
        boolean isOwner = item.getCreatedBy().getId().equals(currentUser.getId());

        if (!isAdmin && !isOwner) {
            throw new UnauthorizedException("You don't have permission to update this item
    | ");
        }

        // Validate status
        Item.Status newStatus;
        try {
            newStatus = Item.Status.valueOf(request.getStatus().toUpperCase());
        } catch (IllegalArgumentException e) {
            throw new BadRequestException("Invalid status. Must be LOST, FOUND, or CLAIMED
    | ");
        }

        // Don't allow changing to CLAIMED unless it's already claimed
        if (newStatus == Item.Status.CLAIMED && item.getStatus() != Item.Status.CLAIMED) {
            throw new BadRequestException("Cannot manually set item status to CLAIMED. Use
    |  the claim endpoint instead.");
        }

```

```
        // Update fields
        item.setName(request.getName().trim());
        item.setDescription(request.getDescription().trim());
        item.setLocation(request.getLocation().trim());
        item.setStatus(newStatus);

        // Update image if provided
        if (image != null && !image.isEmpty()) {
            // Delete old image
            if (item.getImage() != null) {
                fileStorageService.deleteFile(item.getImage());
            }
            String imagePath = fileStorageService.storeFile(image);
            item.setImage(imagePath);
        }

        item = itemRepository.save(item);
        logger.info("User ID {} updated item: {}", user.getId(), item.getId());

        return mapToItemResponse(item);
    }

    @Transactional
    public void deleteItem(Long itemId, UserPrincipal currentUser) {
        if (currentUser == null) {
            throw new UnauthorizedException("You must be logged in to delete an item");
        }

        Item item = itemRepository.findById(itemId)
                .orElseThrow(() -> new ResourceNotFoundException("Item", "id", itemId));

        User user = userRepository.findById(currentUser.getId())
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
    | .getId()));

        // Check permissions
        boolean isAdmin = user.getRole() == User.Role.ADMIN;
        boolean isOwner = item.getCreatedBy().getId().equals(currentUser.getId());

        if (!isAdmin && !isOwner) {
            throw new UnauthorizedException("You don't have permission to delete this item
    | ");
        }

        // Delete related claims
        claimRepository.deleteAll(claimRepository.findByItem(item));

        // Delete related messages
        messageRepository.deleteAll(messageRepository.findAll().stream()
                .filter(m -> m.getItem().getId().equals(itemId))
                .collect(Collectors.toList()));

        // Delete image file if exists
        if (item.getImage() != null) {
            fileStorageService.deleteFile(item.getImage());
        }

        itemRepository.delete(item);
        logger.info("User ID {} deleted item: {}", user.getId(), itemId);
    }

    @Transactional
    public void deleteItemAsAdmin(Long itemId, UserPrincipal currentUser) {
        if (currentUser == null) {
            throw new UnauthorizedException("You must be logged in");
        }

        User user = userRepository.findById(currentUser.getId())
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
    | .getId()));

        // Verify user is admin
        if (user.getRole() != User.Role.ADMIN) {
            throw new UnauthorizedException("Only admins can perform this action");
        }

        Item item = itemRepository.findById(itemId)
                .orElseThrow(() -> new ResourceNotFoundException("Item", "id", itemId));

        // Delete related claims
        claimRepository.deleteAll(claimRepository.findByItem(item));

        // Delete related messages
        messageRepository.deleteAll(messageRepository.findAll().stream()
                .filter(m -> m.getItem().getId().equals(itemId))
```

```
262:                 .collect(Collectors.toList()));
263:
264:         // Delete image file if exists
265:         if (item.getImage() != null) {
266:             fileStorageService.deleteFile(item.getImage());
267:         }
268:
269:         itemRepository.delete(item);
270:         logger.info("Admin ID {} deleted item: {}", user.getId(), itemId);
271:     }
272:
273:     private ItemResponse mapToItemResponse(Item item) {
274:         return ItemResponse.builder()
275:                 .id(item.getId())
276:                 .name(item.getName())
277:                 .description(item.getDescription())
278:                 .location(item.getLocation())
279:                 .status(item.getStatus().name())
280:                 .image(item.getImage())
281:                 .createdBy(item.getCreatedBy().getId())
282:                 .creatorName(item.getCreatedBy().getName())
283:                 .createdAt(item.getCreatedAt())
284:                 .build();
285:     }
286: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\service\MessageService.java

```
  1: package com.lostandfound.service;
  2:
  3: import com.lostandfound.dto.request.MessageRequest;
  4: import com.lostandfound.dto.response.ApiResponse;
  5: import com.lostandfound.dto.response.MessageResponse;
  6: import com.lostandfound.exception.BadRequestException;
  7: import com.lostandfound.exception.ResourceNotFoundException;
  8: import com.lostandfound.exception.UnauthorizedException;
  9: import com.lostandfound.model.Item;
 10: import com.lostandfound.model.Message;
 11: import com.lostandfound.model.User;
 12: import com.lostandfound.repository.ItemRepository;
 13: import com.lostandfound.repository.MessageRepository;
 14: import com.lostandfound.repository.UserRepository;
 15: import com.lostandfound.security.UserPrincipal;
 16: import lombok.RequiredArgsConstructor;
 17: import org.slf4j.Logger;
 18: import org.slf4j.LoggerFactory;
 19: import org.springframework.stereotype.Service;
 20: import org.springframework.transaction.annotation.Transactional;
 21:
 22: import java.util.List;
 23: import java.util.stream.Collectors;
 24:
 25: @Service
 26: @RequiredArgsConstructor
 27: public class MessageService {
 28:
 29:     private static final Logger logger = LoggerFactory.getLogger(MessageService.class);
 30:
 31:     private final MessageRepository messageRepository;
 32:     private final UserRepository userRepository;
 33:     private final ItemRepository itemRepository;
 34:
 35:     @Transactional
 36:     public ApiResponse sendMessage(MessageRequest request, UserPrincipal currentUser) {
 37:         if (currentUser == null) {
 38:             throw new UnauthorizedException("You must be logged in to send a message");
 39:         }
 40:
 41:         User sender = userRepository.findById(currentUser.getId())
 42:                 .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
    | .getId()));
 43:
 44:         User receiver = userRepository.findById(request.getReceiverId())
 45:                 .orElseThrow(() -> new ResourceNotFoundException("Receiver user", "id", re
    | quest.getReceiverId()));
 46:
 47:         Item item = itemRepository.findById(request.getItemId())
 48:                 .orElseThrow(() -> new ResourceNotFoundException("Item", "id", request.get
    | ItemId()));
 49:
 50:         // Validate sender is not sending to themselves
 51:         if (sender.getId().equals(receiver.getId())) {
 52:             throw new BadRequestException("You cannot send a message to yourself");
```

```
        }

        // Validate message content
        if (request.getMessage() == null || request.getMessage().trim().isEmpty()) {
            throw new BadRequestException("Message content cannot be empty");
        }

        if (request.getMessage().trim().length() > 1000) {
            throw new BadRequestException("Message content cannot exceed 1000 characters");
        }

        Message message = new Message();
        message.setSender(sender);
        message.setReceiver(receiver);
        message.setItem(item);
        message.setMessage(request.getMessage().trim());

        messageRepository.save(message);
        logger.info("User {} sent message to user {} about item {}",
                sender.getEmail(), receiver.getEmail(), item.getId());

        return ApiResponse.builder()
                .success(true)
                .message("Message sent successfully")
                .build();
    }

    @Transactional(readOnly = true)
    public List<MessageResponse> getUserMessages(UserPrincipal currentUser) {
        if (currentUser == null) {
            throw new UnauthorizedException("You must be logged in to view messages");
        }

        User user = userRepository.findById(currentUser.getId())
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser.getId()));

        List<Message> messages = messageRepository.findByReceiverOrderBySentAtDesc(user);
        return messages.stream()
                .map(this::mapToMessageResponse)
                .collect(Collectors.toList());
    }

    @Transactional(readOnly = true)
    public List<MessageResponse> getSentMessages(UserPrincipal currentUser) {
        if (currentUser == null) {
            throw new UnauthorizedException("You must be logged in to view sent messages");
        }

        User user = userRepository.findById(currentUser.getId())
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser.getId()));

        List<Message> messages = messageRepository.findBySenderOrderBySentAtDesc(user);
        return messages.stream()
                .map(this::mapToMessageResponse)
                .collect(Collectors.toList());
    }

    @Transactional
    public void deleteMessage(Long messageId, UserPrincipal currentUser) {
        if (currentUser == null) {
            throw new UnauthorizedException("You must be logged in to delete a message");
        }

        Message message = messageRepository.findById(messageId)
                .orElseThrow(() -> new ResourceNotFoundException("Message", "id", messageId));

        User user = userRepository.findById(currentUser.getId())
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser.getId()));

        // Check if user is admin, sender, or receiver
        boolean isAdmin = user.getRole() == User.Role.ADMIN;
        boolean isSender = message.getSender().getId().equals(currentUser.getId());
        boolean isReceiver = message.getReceiver().getId().equals(currentUser.getId());

        if (!isAdmin && !isSender && !isReceiver) {
            throw new UnauthorizedException("You don't have permission to delete this message");
        }

```

```
131:         messageRepository.delete(message);
132:         logger.info("User {} deleted message {}", user.getEmail(), messageId);
133:     }
134:
135:     private MessageResponse mapToMessageResponse(Message message) {
136:         return MessageResponse.builder()
137:                 .id(message.getId())
138:                 .senderId(message.getSender().getId())
139:                 .senderName(message.getSender().getName())
140:                 .receiverId(message.getReceiver().getId())
141:                 .itemId(message.getItem().getId())
142:                 .itemName(message.getItem().getName())
143:                 .message(message.getMessage())
144:                 .sentAt(message.getSentAt())
145:                 .build();
146:     }
147: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\service\RefreshTokenService.java

```
  1: package com.lostandfound.service;
  2:
  3: import com.lostandfound.exception.BadRequestException;
  4: import com.lostandfound.exception.UnauthorizedException;
  5: import com.lostandfound.model.RefreshToken;
  6: import com.lostandfound.model.User;
  7: import com.lostandfound.repository.RefreshTokenRepository;
  8: import com.lostandfound.repository.UserRepository;
  9: import lombok.RequiredArgsConstructor;
 10: import org.slf4j.Logger;
 11: import org.slf4j.LoggerFactory;
 12: import org.springframework.beans.factory.annotation.Value;
 13: import org.springframework.stereotype.Service;
 14: import org.springframework.transaction.annotation.Transactional;
 15:
 16: import java.time.Instant;
 17: import java.util.Optional;
 18: import java.util.UUID;
 19:
 20: @Service
 21: @RequiredArgsConstructor
 22: public class RefreshTokenService {
 23:
 24:     private static final Logger logger = LoggerFactory.getLogger(RefreshTokenService.class
    | );
 25:
 26:     private final RefreshTokenRepository refreshTokenRepository;
 27:     private final UserRepository userRepository;
 28:
 29:     @Value("${jwt.refresh.expiration}")
 30:     private Long refreshTokenDurationMs;
 31:
 32:     @Transactional
 33:     public RefreshToken createRefreshToken(Long userId, String ipAddress, String userAgent
    | ) {
 34:         User user = userRepository.findById(userId)
 35:                 .orElseThrow(() -> new BadRequestException("User not found"));
 36:
 37:         // Delete old refresh tokens for this user
 38:         refreshTokenRepository.deleteByUser(user);
 39:
 40:         RefreshToken refreshToken = RefreshToken.builder()
 41:                 .user(user)
 42:                 .token(UUID.randomUUID().toString())
 43:                 .expiryDate(Instant.now().plusMillis(refreshTokenDurationMs))
 44:                 .revoked(false)
 45:                 .ipAddress(ipAddress)
 46:                 .userAgent(userAgent)
 47:                 .build();
 48:
 49:         logger.info("Created new refresh token for user: {}", user.getEmail());
 50:         return refreshTokenRepository.save(refreshToken);
 51:     }
 52:
 53:     @Transactional(readOnly = true)
 54:     public Optional<RefreshToken> findByToken(String token) {
 55:         if (token == null || token.trim().isEmpty()) {
 56:             return Optional.empty();
 57:         }
 58:         return refreshTokenRepository.findByToken(token.trim());
 59:     }
 60:
 61:     @Transactional
```

```
 62:     public RefreshToken verifyExpiration(RefreshToken token) {
 63:         if (token.getExpiryDate().compareTo(Instant.now()) < 0) {
 64:             refreshTokenRepository.delete(token);
 65:             logger.warn("Refresh token expired for user: {}", token.getUser().getEmail());
 66:             throw new BadRequestException("Refresh token expired. Please login again.");
 67:         }
 68:
 69:         if (token.isRevoked()) {
 70:             logger.warn("Attempted to use revoked refresh token for user: {}", token.getUs
    | er().getEmail());
 71:             throw new BadRequestException("Refresh token has been revoked. Please login ag
    | ain.");
 72:         }
 73:
 74:         return token;
 75:     }
 76:
 77:     @Transactional
 78:     public void revokeToken(String token) {
 79:         if (token == null || token.trim().isEmpty()) {
 80:             throw new BadRequestException("Invalid refresh token");
 81:         }
 82:
 83:         RefreshToken refreshToken = refreshTokenRepository.findByToken(token.trim())
 84:                 .orElseThrow(() -> new BadRequestException("Invalid refresh token"));
 85:
 86:         if (refreshToken.isRevoked()) {
 87:             throw new BadRequestException("Token is already revoked");
 88:         }
 89:
 90:         refreshToken.setRevoked(true);
 91:         refreshTokenRepository.save(refreshToken);
 92:         logger.info("Revoked refresh token for user: {}", refreshToken.getUser().getEmail(
    | ));
 93:     }
 94:
 95:     @Transactional
 96:     public void revokeTokenForUser(String token, Long userId) {
 97:         if (token == null || token.trim().isEmpty()) {
 98:             throw new BadRequestException("Invalid refresh token");
 99:         }
100:
101:         RefreshToken refreshToken = refreshTokenRepository.findByToken(token.trim())
102:                 .orElseThrow(() -> new BadRequestException("Invalid refresh token"));
103:
104:         // Verify the token belongs to the user
105:         if (!refreshToken.getUser().getId().equals(userId)) {
106:             logger.warn("User {} attempted to revoke token belonging to user {}",
107:                     userId, refreshToken.getUser().getId());
108:             throw new UnauthorizedException("This token does not belong to you");
109:         }
110:
111:         if (refreshToken.isRevoked()) {
112:             throw new BadRequestException("Token is already revoked");
113:         }
114:
115:         refreshToken.setRevoked(true);
116:         refreshTokenRepository.save(refreshToken);
117:         logger.info("Revoked refresh token for user: {}", refreshToken.getUser().getEmail(
    | ));
118:     }
119:
120:     @Transactional
121:     public void revokeAllUserTokens(Long userId) {
122:         User user = userRepository.findById(userId)
123:                 .orElseThrow(() -> new BadRequestException("User not found"));
124:
125:         refreshTokenRepository.deleteByUser(user);
126:         logger.info("Revoked all refresh tokens for user: {}", user.getEmail());
127:     }
128:
129:     @Transactional
130:     public void deleteExpiredTokens() {
131:         int deletedCount = refreshTokenRepository.deleteAllExpiredTokens(Instant.now());
132:         logger.info("Deleted {} expired refresh tokens", deletedCount);
133:     }
134: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\service\UserService.java

```
  1: package com.lostandfound.service;
  2:
  3: import com.lostandfound.dto.request.LoginRequest;
```

```
 4: import com.lostandfound.dto.request.RegisterRequest;
 5: import com.lostandfound.dto.response.AuthResponse;
 6: import com.lostandfound.dto.response.TokenRefreshResponse;
 7: import com.lostandfound.exception.BadRequestException;
 8: import com.lostandfound.exception.ResourceNotFoundException;
 9: import com.lostandfound.model.RefreshToken;
10: import com.lostandfound.model.User;
11: import com.lostandfound.repository.UserRepository;
12: import com.lostandfound.security.JwtTokenProvider;
13: import com.lostandfound.security.UserPrincipal;
14: import lombok.RequiredArgsConstructor;
15: import org.slf4j.Logger;
16: import org.slf4j.LoggerFactory;
17: import org.springframework.beans.factory.annotation.Value;
18: import org.springframework.security.authentication.AuthenticationManager;
19: import org.springframework.security.authentication.BadCredentialsException;
20: import org.springframework.security.authentication.LockedException;
21: import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
22: import org.springframework.security.core.Authentication;
23: import org.springframework.security.core.context.SecurityContextHolder;
24: import org.springframework.security.crypto.password.PasswordEncoder;
25: import org.springframework.stereotype.Service;
26: import org.springframework.transaction.annotation.Transactional;
27:
28: @Service
29: @RequiredArgsConstructor
30: public class UserService {
31:
32:     private static final Logger logger = LoggerFactory.getLogger(UserService.class);
33:
34:     private final UserRepository userRepository;
35:     private final PasswordEncoder passwordEncoder;
36:     private final AuthenticationManager authenticationManager;
37:     private final JwtTokenProvider tokenProvider;
38:     private final RefreshTokenService refreshTokenService;
39:
40:     @Value("${jwt.expiration}")
41:     private Long jwtExpirationMs;
42:
43:     @Transactional
44:     public AuthResponse registerUser(RegisterRequest request, String ipAddress, String use
   | rAgent) {
45:         // Normalize email to lowercase
46:         String email = request.getEmail().toLowerCase().trim();
47:
48:         // Check if email already exists
49:         if (userRepository.existsByEmail(email)) {
50:             logger.warn("Registration attempt with existing email");
51:             // Use generic message to prevent user enumeration
52:             // Add a small delay to prevent timing attacks
53:             try {
54:                 Thread.sleep(100 + (long)(Math.random() * 200)); // 100-300ms random delay
55:             } catch (InterruptedException e) {
56:                 Thread.currentThread().interrupt();
57:             }
58:             throw new BadRequestException("Registration failed. Please check your informat
   | ion and try again.");
59:         }
60:
61:         // Validate password confirmation
62:         if (!request.getPassword().equals(request.getConfirmPassword())) {
63:             throw new BadRequestException("Passwords do not match");
64:         }
65:
66:         // Create new user
67:         User user = new User();
68:         user.setName(request.getName().trim());
69:         user.setEmail(email);
70:         user.setPassword(passwordEncoder.encode(request.getPassword()));
71:         user.setRole(User.Role.USER);
72:
73:         try {
74:             user = userRepository.save(user);
75:             logger.info("New user registered successfully with ID: {}", user.getId());
76:         } catch (Exception e) {
77:             logger.error("Error during user registration", e);
78:             throw new BadRequestException("Registration failed. Please try again.");
79:         }
80:
81:         // Auto-login after registration
82:         Authentication authentication = authenticationManager.authenticate(
83:                 new UsernamePasswordAuthenticationToken(email, request.getPassword())
84:         );
85:
86:         SecurityContextHolder.getContext().setAuthentication(authentication);
```

```
 87:
 88:         // Generate access token
 89:         String accessToken = tokenProvider.generateToken(authentication);
 90:
 91:         // Generate refresh token
 92:         RefreshToken refreshToken = refreshTokenService.createRefreshToken(
 93:                 user.getId(), ipAddress, userAgent
 94:         );
 95:
 96:         // Build user DTO
 97:         AuthResponse.UserDTO userDTO = AuthResponse.UserDTO.builder()
 98:                 .id(user.getId())
 99:                 .name(user.getName())
100:                 .email(user.getEmail())
101:                 .role(user.getRole().name())
102:                 .build();
103:
104:         return AuthResponse.builder()
105:                 .success(true)
106:                 .message("Registration successful")
107:                 .accessToken(accessToken)
108:                 .refreshToken(refreshToken.getToken())
109:                 .tokenType("Bearer")
110:                 .expiresIn(jwtExpirationMs / 1000) // Convert to seconds
111:                 .user(userDTO)
112:                 .build();
113:     }
114:
115:     @Transactional
116:     public AuthResponse loginUser(LoginRequest request, String ipAddress, String userAgent
    | ) {
117:         String email = request.getEmail().toLowerCase().trim();
118:
119:         try {
120:             // Authenticate user
121:             Authentication authentication = authenticationManager.authenticate(
122:                     new UsernamePasswordAuthenticationToken(email, request.getPassword())
123:             );
124:
125:             SecurityContextHolder.getContext().setAuthentication(authentication);
126:
127:             // Generate access token
128:             String accessToken = tokenProvider.generateToken(authentication);
129:
130:             // Fetch user details
131:             User user = userRepository.findByEmail(email)
132:                     .orElseThrow(() -> new BadRequestException("User not found"));
133:
134:             // Generate refresh token
135:             RefreshToken refreshToken = refreshTokenService.createRefreshToken(
136:                     user.getId(), ipAddress, userAgent
137:             );
138:
139:             logger.info("User logged in successfully with ID: {}", user.getId());
140:
141:             // Build user DTO
142:             AuthResponse.UserDTO userDTO = AuthResponse.UserDTO.builder()
143:                     .id(user.getId())
144:                     .name(user.getName())
145:                     .email(user.getEmail())
146:                     .role(user.getRole().name())
147:                     .build();
148:
149:             return AuthResponse.builder()
150:                     .success(true)
151:                     .message("Login successful")
152:                     .accessToken(accessToken)
153:                     .refreshToken(refreshToken.getToken())
154:                     .tokenType("Bearer")
155:                     .expiresIn(jwtExpirationMs / 1000) // Convert to seconds
156:                     .user(userDTO)
157:                     .build();
158:
159:         } catch (BadCredentialsException e) {
160:             logger.warn("Failed login attempt");
161:             throw new BadCredentialsException("Invalid email or password");
162:         } catch (LockedException e) {
163:             logger.warn("Login attempt for locked account");
164:             throw new BadRequestException("Account is locked. Please contact support.");
165:         } catch (Exception e) {
166:             logger.error("Error during login", e);
167:             throw new BadRequestException("Login failed. Please try again.");
168:         }
169:     }
170:
```

```
171:     @Transactional
172:     public void logoutUser(Long userId) {
173:         User user = userRepository.findById(userId)
174:                 .orElseThrow(() -> new BadRequestException("User not found"));
175:
176:         // Delete all refresh tokens for this user
177:         refreshTokenService.revokeAllUserTokens(userId);
178:
179:         logger.info("User logged out successfully with ID: {}", user.getId());
180:     }
181:
182:     @Transactional
183:     public TokenRefreshResponse refreshToken(String refreshTokenStr) {
184:         return refreshTokenService.findByToken(refreshTokenStr)
185:                 .map(refreshTokenService::verifyExpiration)
186:                 .map(RefreshToken::getUser)
187:                 .map(user -> {
188:                     UserPrincipal userPrincipal = UserPrincipal.create(user);
189:                     Authentication auth = new UsernamePasswordAuthenticationToken(
190:                             userPrincipal, null, userPrincipal.getAuthorities()
191:                     );
192:
193:                     String newAccessToken = tokenProvider.generateToken(auth);
194:
195:                     return TokenRefreshResponse.builder()
196:                             .success(true)
197:                             .message("Token refreshed successfully")
198:                             .accessToken(newAccessToken)
199:                             .refreshToken(refreshTokenStr)
200:                             .tokenType("Bearer")
201:                             .build();
202:                 })
203:                 .orElseThrow(() -> new BadRequestException("Invalid refresh token"));
204:     }
205:
206:     @Transactional(readOnly = true)
207:     public User getUserById(Long userId) {
208:         return userRepository.findById(userId)
209:                 .orElseThrow(() -> new BadRequestException("User not found"));
210:     }
211:
212:     @Transactional(readOnly = true)
213:     public User getUserByEmail(String email) {
214:         return userRepository.findByEmail(email.toLowerCase().trim())
215:                 .orElseThrow(() -> new BadRequestException("User not found"));
216:     }
217:
218:     @Transactional
219:     public void deleteUser(Long userId, UserPrincipal currentUser) {
220:         // Verify current user is admin
221:         User adminUser = userRepository.findById(currentUser.getId())
222:                 .orElseThrow(() -> new BadRequestException("Admin user not found"));
223:
224:         if (adminUser.getRole() != User.Role.ADMIN) {
225:             throw new BadRequestException("Only admins can delete users");
226:         }
227:
228:         // Find the user to delete
229:         User userToDelete = userRepository.findById(userId)
230:                 .orElseThrow(() -> new ResourceNotFoundException("User", "id", userId));
231:
232:         // Prevent deleting admin users
233:         if (userToDelete.getRole() == User.Role.ADMIN) {
234:             throw new BadRequestException("Cannot delete admin user");
235:         }
236:
237:         logger.info("Admin ID {} deleting user ID {}", adminUser.getId(), userToDelete.get
    | Id());
238:
239:         // Delete all refresh tokens for this user
240:         refreshTokenService.revokeAllUserTokens(userId);
241:
242:         // Delete the user (cascading will handle related data if configured)
243:         userRepository.delete(userToDelete);
244:
245:         logger.info("Successfully deleted user ID: {}", userToDelete.getId());
246:     }
247: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\util\CookieUtil.java

```
  1: package com.lostandfound.util;
  2:
```

```java
import jakarta.servlet.http.Cookie;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;

import java.util.Arrays;
import java.util.Optional;

@Component
public class CookieUtil {

    @Value("${cookie.domain:localhost}")
    private String cookieDomain;

    @Value("${cookie.secure:false}")
    private boolean cookieSecure;

    @Value("${cookie.same-site:Lax}")
    private String cookieSameSite;

    // Cookie names
    public static final String ACCESS_TOKEN_COOKIE = "accessToken";
    public static final String REFRESH_TOKEN_COOKIE = "refreshToken";

    // Cookie max ages (in seconds)
    private static final int ACCESS_TOKEN_MAX_AGE = 15 * 60; // 15 minutes
    private static final int REFRESH_TOKEN_MAX_AGE = 7 * 24 * 60 * 60; // 7 days

    /**
     * Add access token cookie
     */
    public void addAccessTokenCookie(HttpServletResponse response, String token) {
        addCookie(response, ACCESS_TOKEN_COOKIE, token, ACCESS_TOKEN_MAX_AGE, true);
    }

    /**
     * Add refresh token cookie
     */
    public void addRefreshTokenCookie(HttpServletResponse response, String token) {
        addCookie(response, REFRESH_TOKEN_COOKIE, token, REFRESH_TOKEN_MAX_AGE, true);
    }

    /**
     * Generic method to add cookie
     */
    private void addCookie(HttpServletResponse response, String name, String value,
                           int maxAge, boolean httpOnly) {
        Cookie cookie = new Cookie(name, value);
        cookie.setPath("/");
        cookie.setHttpOnly(httpOnly);
        cookie.setMaxAge(maxAge);
        cookie.setSecure(cookieSecure); // Set to true in production with HTTPS

        // SameSite attribute (Lax, Strict, None)
        // Note: SameSite=None requires Secure=true
        String cookieHeader = String.format(
            "%s=%s; Path=/; Max-Age=%d; HttpOnly=%s; Secure=%s; SameSite=%s",
            name, value, maxAge,
            httpOnly ? "true" : "false",
            cookieSecure ? "true" : "false",
            cookieSameSite
        );

        response.addHeader("Set-Cookie", cookieHeader);
    }

    /**
     * Get cookie value by name
     */
    public Optional<String> getCookie(HttpServletRequest request, String name) {
        if (request.getCookies() == null) {
            return Optional.empty();
        }

        return Arrays.stream(request.getCookies())
                .filter(cookie -> name.equals(cookie.getName()))
                .map(Cookie::getValue)
                .findFirst();
    }

    /**
     * Get access token from cookie
     */
    public Optional<String> getAccessToken(HttpServletRequest request) {
```

```
 88:         return getCookie(request, ACCESS_TOKEN_COOKIE);
 89:     }
 90:
 91:     /**
 92:      * Get refresh token from cookie
 93:      */
 94:     public Optional<String> getRefreshToken(HttpServletRequest request) {
 95:         return getCookie(request, REFRESH_TOKEN_COOKIE);
 96:     }
 97:
 98:     /**
 99:      * Delete cookie
100:      */
101:     public void deleteCookie(HttpServletResponse response, String name) {
102:         Cookie cookie = new Cookie(name, null);
103:         cookie.setPath("/");
104:         cookie.setHttpOnly(true);
105:         cookie.setMaxAge(0);
106:         cookie.setSecure(cookieSecure);
107:
108:         response.addCookie(cookie);
109:     }
110:
111:     /**
112:      * Delete access token cookie
113:      */
114:     public void deleteAccessToken(HttpServletResponse response) {
115:         deleteCookie(response, ACCESS_TOKEN_COOKIE);
116:     }
117:
118:     /**
119:      * Delete refresh token cookie
120:      */
121:     public void deleteRefreshToken(HttpServletResponse response) {
122:         deleteCookie(response, REFRESH_TOKEN_COOKIE);
123:     }
124:
125:     /**
126:      * Delete all auth cookies
127:      */
128:     public void deleteAllAuthCookies(HttpServletResponse response) {
129:         deleteAccessToken(response);
130:         deleteRefreshToken(response);
131:     }
132: }
```
