■ 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 + " seconds."
 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:         if (xfHeader == null || xfHeader.isEmpty()) {
 71:             return request.getRemoteAddr();
 72:         }
 73:         return xfHeader.split(",")[0].trim();
 74:     }
 75: }
```

■ 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;
```

```
import io.github.bucket4j.Bucket;
import io.github.bucket4j.Refill;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import java.time.Duration;
import java.util.Map;
import java.util.concurrent.ConcurrentHashMap;

@Configuration
public class RateLimitConfig {

    /**
     * In-memory cache for storing buckets per IP address
     * For production, consider using Redis or Hazelcast for distributed caching
     */
    private final Map<String, Bucket> cache = new ConcurrentHashMap<>();

    /**
     * Rate limit configurations for different endpoints
     */
    public enum RateLimitType {
        // Auth endpoints - 5 requests per minute
        AUTH(5, Duration.ofMinutes(1)),

        // General API endpoints - 100 requests per minute
        API(100, Duration.ofMinutes(1)),

        // Admin endpoints - 50 requests per minute
        ADMIN(50, Duration.ofMinutes(1)),

        // File upload - 10 requests per 5 minutes
        UPLOAD(10, Duration.ofMinutes(5)),

        // Public endpoints - 200 requests per minute
        PUBLIC(200, Duration.ofMinutes(1));

        private final long capacity;
        private final Duration refillDuration;

        RateLimitType(long capacity, Duration refillDuration) {
            this.capacity = capacity;
            this.refillDuration = refillDuration;
        }

        public long getCapacity() {
            return capacity;
        }

        public Duration getRefillDuration() {
            return refillDuration;
        }
    }

    /**
     * Resolve bucket for given key and rate limit type
     */
    public Bucket resolveBucket(String key, RateLimitType limitType) {
        return cache.computeIfAbsent(key, k -> createNewBucket(limitType));
    }

    /**
     * Create new bucket with specified rate limit
     */
    private Bucket createNewBucket(RateLimitType limitType) {
        Bandwidth limit = Bandwidth.classic(
                limitType.getCapacity(),
                Refill.intervally(limitType.getCapacity(), limitType.getRefillDuration())
        );
        return Bucket.builder()
                .addLimit(limit)
                .build();
    }

    /**
     * Clear bucket for specific key (useful for testing or admin actions)
     */
    public void clearBucket(String key) {
        cache.remove(key);
    }

    /**
     * Clear all buckets (useful for testing)
     */
    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.XXssProtectionHeaderWriter;
 25: import org.springframework.web.cors.CorsConfiguration;
 26: import org.springframework.web.cors.CorsConfigurationSource;
 27: import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
 28:
 29: import java.util.Arrays;
 30: import java.util.List;
 31:
 32: @Configuration
 33: @EnableWebSecurity
 34: @EnableMethodSecurity(prePostEnabled = true, securedEnabled = true)
 35: @RequiredArgsConstructor
 36: public class SecurityConfig {
 37:
 38:     private final JwtAuthenticationFilter jwtAuthenticationFilter;
 39:     private final RateLimitFilter rateLimitFilter;
 40:     private final UserDetailsService userDetailsService;
 41:
 42:     @Value("${cors.allowed-origins}")
 43:     private String allowedOrigins;
 44:
 45:     @Value("${security.csrf.enabled:false}")
 46:     private boolean csrfEnabled;
 47:
 48:     @Bean
 49:     public PasswordEncoder passwordEncoder() {
 50:         // Use strength 12 for production (default is 10)
 51:         return new BCryptPasswordEncoder(12);
 52:     }
 53:
 54:     @Bean
 55:     public DaoAuthenticationProvider authenticationProvider() {
 56:         DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
 57:         authProvider.setUserDetailsService(userDetailsService);
 58:         authProvider.setPasswordEncoder(passwordEncoder());
 59:         authProvider.setHideUserNotFoundExceptions(true); // Security best practice
 60:         return authProvider;
 61:     }
 62:
 63:     @Bean
 64:     public AuthenticationManager authenticationManager(AuthenticationConfiguration authCon
    | fig)
 65:             throws Exception {
```

```
66:         return authConfig.getAuthenticationManager();
67:     }
68:
69:     @Bean
70:     public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
71:         // CSRF Token Handler for SPA
72:         CsrfTokenRequestAttributeHandler requestHandler = new CsrfTokenRequestAttributeHan
    | dler();
73:         requestHandler.setCsrfRequestAttributeName("_csrf");
74:
75:         http
76:                 // CSRF Configuration
77:                 .csrf(csrf -> {
78:                     if (csrfEnabled) {
79:                         // Enable CSRF with Cookie-based tokens for SPA
80:                         csrf.csrfTokenRepository(CookieCsrfTokenRepository.withHttpOnlyFal
    | se())
81:                                 .csrfTokenRequestHandler(requestHandler)
82:                                 .ignoringRequestMatchers("/api/auth/**"); // Exclude auth
    | endpoints
83:                     } else {
84:                         // Disable CSRF for stateless JWT authentication
85:                         csrf.disable();
86:                     }
87:                 })
88:
89:                 // CORS Configuration
90:                 .cors(cors -> cors.configurationSource(corsConfigurationSource()))
91:
92:                 // Session Management - Stateless
93:                 .sessionManagement(session ->
94:                         session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
95:
96:                 // Authorization Rules
97:                 .authorizeHttpRequests(auth -> auth
98:                         // Public endpoints
99:                         .requestMatchers(
100:                                "/api/auth/**",
101:                                "/uploads/**",
102:                                "/static/**",
103:                                "/error",
104:                                "/actuator/health",
105:                                "/actuator/info"
106:                        ).permitAll()
107:
108:                        // Admin endpoints
109:                        .requestMatchers("/admin/**").hasRole("ADMIN")
110:
111:                        // All other requests require authentication
112:                        .anyRequest().authenticated()
113:                )
114:
115:                // Authentication Provider
116:                .authenticationProvider(authenticationProvider())
117:
118:                // Add filters in correct order:
119:                // 1. First add JWT filter before UsernamePasswordAuthenticationFilter
120:                .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFi
    | lter.class)
121:                // 2. Then add Rate Limit filter before JWT filter
122:                .addFilterBefore(rateLimitFilter, JwtAuthenticationFilter.class)
123:
124:                // Security Headers
125:                .headers(headers -> headers
126:                        // Content Security Policy
127:                        .contentSecurityPolicy(csp ->
128:                                csp.policyDirectives("default-src 'self'; " +
129:                                        "script-src 'self' 'unsafe-inline'; " +
130:                                        "style-src 'self' 'unsafe-inline'; " +
131:                                        "img-src 'self' data: https:; " +
132:                                        "font-src 'self' data:; " +
133:                                        "connect-src 'self'"))
134:
135:                        // Frame Options - Prevent Clickjacking
136:                        .frameOptions(frame -> frame.deny())
137:
138:                        // XSS Protection
139:                        .xssProtection(xss -> xss
140:                                .headerValue(XXssProtectionHeaderWriter.HeaderValue.ENABLE
    | D_MODE_BLOCK))
141:
142:                        // HSTS - Force HTTPS
143:                        .httpStrictTransportSecurity(hsts -> hsts
144:                                .includeSubDomains(true)
145:                                .maxAgeInSeconds(31536000)) // 1 year
```

```
146:
147:                         // Prevent MIME Sniffing
148:                         .contentTypeOptions(contentType -> {})
149:                 );
150:
151:         return http.build();
152:     }
153:
154:     @Bean
155:     public CorsConfigurationSource corsConfigurationSource() {
156:         CorsConfiguration configuration = new CorsConfiguration();
157:
158:         // Parse allowed origins from application.properties
159:         List<String> origins = Arrays.asList(allowedOrigins.split(","));
160:         configuration.setAllowedOriginPatterns(origins);
161:
162:         // Allowed HTTP methods
163:         configuration.setAllowedMethods(Arrays.asList(
164:                 "GET", "POST", "PUT", "DELETE", "PATCH", "OPTIONS"
165:         ));
166:
167:         // Allowed headers
168:         configuration.setAllowedHeaders(Arrays.asList(
169:                 "Authorization",
170:                 "Content-Type",
171:                 "X-Requested-With",
172:                 "Accept",
173:                 "Origin",
174:                 "Access-Control-Request-Method",
175:                 "Access-Control-Request-Headers",
176:                 "X-CSRF-TOKEN"
177:         ));
178:
179:         // Exposed headers (so frontend can read them)
180:         configuration.setExposedHeaders(Arrays.asList(
181:                 "Authorization",
182:                 "X-CSRF-TOKEN",
183:                 "Access-Control-Allow-Origin",
184:                 "Access-Control-Allow-Credentials"
185:         ));
186:
187:         // Allow credentials (cookies, authorization headers)
188:         configuration.setAllowCredentials(true);
189:
190:         // Cache preflight response for 1 hour
191:         configuration.setMaxAge(3600L);
192:
193:         UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
194:         source.registerCorsConfiguration("/**", configuration);
195:
196:         return source;
197:     }
198: }
```

■ 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:
```

```
    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/uploads/**")
                .addResourceLocations("file:" + uploadDir + "/");
        registry.addResourceHandler("/static/**")
                .addResourceLocations("file:" + uploadDir + "/");
    }
}
```

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

```
package com.lostandfound.controller;

import com.lostandfound.dto.response.ApiResponse;
import com.lostandfound.dto.response.ClaimResponse;
import com.lostandfound.dto.response.FeedbackResponse;
import com.lostandfound.dto.response.ItemResponse;
import com.lostandfound.exception.ResourceNotFoundException;
import com.lostandfound.model.User;
import com.lostandfound.repository.UserRepository;
import com.lostandfound.security.UserPrincipal;
import com.lostandfound.service.ClaimService;
import com.lostandfound.service.FeedbackService;
import com.lostandfound.service.ItemService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
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;
import java.util.stream.Collectors;

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

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

    @GetMapping("/dashboard")
    public ResponseEntity<Map<String, Object>> getAdminDashboard() {
        List<ItemResponse> items = itemService.searchItems("", "");
        List<ClaimResponse> claims = claimService.getUserClaims(null); // Get all claims
        List<FeedbackResponse> feedback = feedbackService.getAllFeedback();
        List<User> users = userRepository.findByRoleNot(User.Role.ADMIN);

        List<Map<String, Object>> userList = users.stream()
                .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("items", items);
        response.put("claims", claims);
        response.put("users", userList);
        response.put("feedback", feedback);

        return ResponseEntity.ok(response);
    }

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

        itemService.deleteItem(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) {
        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) {
        User user = userRepository.findById(userId)
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", userId));

        if (user.getRole() == User.Role.ADMIN) {
            throw new ResourceNotFoundException("Cannot delete admin user");
        }

        userRepository.delete(user);

        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) {
        feedbackService.deleteFeedback(feedbackId);

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

        return ResponseEntity.ok(response);
    }
}
```

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

```
package com.lostandfound.controller;

import com.lostandfound.dto.request.LoginRequest;
import com.lostandfound.dto.request.RegisterRequest;
import com.lostandfound.dto.request.TokenRefreshRequest;
import com.lostandfound.dto.response.ApiResponse;
import com.lostandfound.dto.response.AuthResponse;
import com.lostandfound.dto.response.TokenRefreshResponse;
import com.lostandfound.security.UserPrincipal;
import com.lostandfound.service.RefreshTokenService;
import com.lostandfound.service.UserService;
import jakarta.servlet.http.HttpServletRequest;
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.*;

@RestController
@RequestMapping("/api/auth")
@RequiredArgsConstructor
public class AuthController {

    private final UserService userService;
    private final RefreshTokenService refreshTokenService;

    @PostMapping("/register")
    public ResponseEntity<AuthResponse> registerUser(
            @Valid @RequestBody RegisterRequest request,
            HttpServletRequest httpRequest) {

```

```
 32:         String ipAddress = getClientIP(httpRequest);
 33:         String userAgent = httpRequest.getHeader("User-Agent");
 34:
 35:         AuthResponse response = userService.registerUser(request, ipAddress, userAgent);
 36:         return ResponseEntity.ok(response);
 37:     }
 38:
 39:     @PostMapping("/login")
 40:     public ResponseEntity<AuthResponse> loginUser(
 41:             @Valid @RequestBody LoginRequest request,
 42:             HttpServletRequest httpRequest) {
 43:
 44:         String ipAddress = getClientIP(httpRequest);
 45:         String userAgent = httpRequest.getHeader("User-Agent");
 46:
 47:         AuthResponse response = userService.loginUser(request, ipAddress, userAgent);
 48:         return ResponseEntity.ok(response);
 49:     }
 50:
 51:     @PostMapping("/refresh")
 52:     public ResponseEntity<TokenRefreshResponse> refreshToken(
 53:             @Valid @RequestBody TokenRefreshRequest request) {
 54:
 55:         TokenRefreshResponse response = userService.refreshToken(request.getRefreshToken()
    | );
 56:         return ResponseEntity.ok(response);
 57:     }
 58:
 59:     @PostMapping("/logout")
 60:     public ResponseEntity<ApiResponse> logout(
 61:             @AuthenticationPrincipal UserPrincipal currentUser,
 62:             @RequestBody(required = false) TokenRefreshRequest request) {
 63:
 64:         if (request != null && request.getRefreshToken() != null) {
 65:             refreshTokenService.revokeToken(request.getRefreshToken());
 66:         }
 67:
 68:         ApiResponse response = ApiResponse.builder()
 69:                 .success(true)
 70:                 .message("Logged out successfully")
 71:                 .build();
 72:
 73:         return ResponseEntity.ok(response);
 74:     }
 75:
 76:     @GetMapping("/validate")
 77:     public ResponseEntity<ApiResponse> validateToken(
 78:             @AuthenticationPrincipal UserPrincipal currentUser) {
 79:
 80:         ApiResponse response = ApiResponse.builder()
 81:                 .success(true)
 82:                 .message("Token is valid")
 83:                 .data(currentUser.getEmail())
 84:                 .build();
 85:
 86:         return ResponseEntity.ok(response);
 87:     }
 88:
 89:     private String getClientIP(HttpServletRequest request) {
 90:         String xfHeader = request.getHeader("X-Forwarded-For");
 91:         if (xfHeader == null) {
 92:             return request.getRemoteAddr();
 93:         }
 94:         return xfHeader.split(",")[0];
 95:     }
 96: }
```

■ 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.security.UserPrincipal;
  5: import com.lostandfound.service.ClaimService;
  6: import lombok.RequiredArgsConstructor;
  7: import org.springframework.http.ResponseEntity;
  8: import org.springframework.security.core.annotation.AuthenticationPrincipal;
  9: import org.springframework.web.bind.annotation.*;
 10:
 11: @RestController
 12: @RequestMapping("/items")
 13: @RequiredArgsConstructor
 14: public class ClaimController {
```

```
 15:
 16:     private final ClaimService claimService;
 17:
 18:     @PostMapping("/{itemId}/claim")
 19:     public ResponseEntity<ApiResponse> claimItem(
 20:             @PathVariable Long itemId,
 21:             @AuthenticationPrincipal UserPrincipal currentUser) {
 22:
 23:         ApiResponse response = claimService.claimItem(itemId, currentUser);
 24:         return ResponseEntity.ok(response);
 25:     }
 26: }
```

■ 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.security.UserPrincipal;
  8: import com.lostandfound.service.ClaimService;
  9: import com.lostandfound.service.ItemService;
 10: import com.lostandfound.service.MessageService;
 11: import lombok.RequiredArgsConstructor;
 12: import org.springframework.http.ResponseEntity;
 13: import org.springframework.security.core.annotation.AuthenticationPrincipal;
 14: import org.springframework.web.bind.annotation.GetMapping;
 15: import org.springframework.web.bind.annotation.RequestMapping;
 16: import org.springframework.web.bind.annotation.RestController;
 17:
 18: import java.util.List;
 19:
 20: @RestController
 21: @RequestMapping("/dashboard")
 22: @RequiredArgsConstructor
 23: public class DashboardController {
 24:
 25:     private final ItemService itemService;
 26:     private final ClaimService claimService;
 27:     private final MessageService messageService;
 28:
 29:     @GetMapping
 30:     public ResponseEntity<DashboardResponse> getDashboard(
 31:             @AuthenticationPrincipal UserPrincipal currentUser) {
 32:
 33:         List<ItemResponse> items = itemService.getUserItems(currentUser);
 34:         List<ClaimResponse> claims = claimService.getUserClaims(currentUser);
 35:         List<MessageResponse> messages = messageService.getUserMessages(currentUser);
 36:
 37:         String role = currentUser.getAuthorities().iterator().next()
 38:                 .getAuthority().replace("ROLE_", "");
 39:
 40:         DashboardResponse.UserInfo userInfo = DashboardResponse.UserInfo.builder()
 41:                 .name(currentUser.getName())
 42:                 .email(currentUser.getEmail())
 43:                 .role(role)
 44:                 .build();
 45:
 46:         DashboardResponse response = DashboardResponse.builder()
 47:                 .user(userInfo)
 48:                 .items(items)
 49:                 .claims(claims)
 50:                 .messages(messages)
 51:                 .build();
 52:
 53:         return ResponseEntity.ok(response);
 54:     }
 55: }
```

■ 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.security.UserPrincipal;
  6: import com.lostandfound.service.FeedbackService;
  7: import jakarta.validation.Valid;
  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: @RestController
14: @RequestMapping("/feedback")
15: @RequiredArgsConstructor
16: public class FeedbackController {
17:
18:     private final FeedbackService feedbackService;
19:
20:     @PostMapping
21:     public ResponseEntity<ApiResponse> submitFeedback(
22:             @Valid @RequestBody FeedbackRequest request,
23:             @AuthenticationPrincipal UserPrincipal currentUser) {
24:
25:         ApiResponse response = feedbackService.submitFeedback(request, currentUser);
26:         return ResponseEntity.ok(response);
27:     }
28: }
```

■ 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.security.UserPrincipal;
  7: import com.lostandfound.service.ItemService;
  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: import org.springframework.web.multipart.MultipartFile;
 14:
 15: import java.util.HashMap;
 16: import java.util.List;
 17: import java.util.Map;
 18:
 19: @RestController
 20: @RequestMapping("/items")
 21: @RequiredArgsConstructor
 22: public class ItemController {
 23:
 24:     private final ItemService itemService;
 25:
 26:     @PostMapping
 27:     public ResponseEntity<ApiResponse> createItem(
 28:             @Valid @ModelAttribute ItemRequest request,
 29:             @RequestParam(value = "image", required = false) MultipartFile image,
 30:             @AuthenticationPrincipal UserPrincipal currentUser) {
 31:
 32:         ItemResponse itemResponse = itemService.createItem(request, image, currentUser);
 33:
 34:         ApiResponse response = ApiResponse.builder()
 35:                 .success(true)
 36:                 .message("Item registered successfully")
 37:                 .data(itemResponse)
 38:                 .build();
 39:
 40:         return ResponseEntity.ok(response);
 41:     }
 42:
 43:     @GetMapping
 44:     public ResponseEntity<Map<String, List<ItemResponse>>> getItems(
 45:             @RequestParam(required = false, defaultValue = "") String search,
 46:             @RequestParam(required = false, defaultValue = "") String status) {
 47:
 48:         List<ItemResponse> items = itemService.searchItems(search, status);
 49:
 50:         Map<String, List<ItemResponse>> response = new HashMap<>();
 51:         response.put("items", items);
 52:
 53:         return ResponseEntity.ok(response);
 54:     }
 55:
 56:     @DeleteMapping("/{itemId}")
 57:     public ResponseEntity<ApiResponse> deleteItem(
 58:             @PathVariable Long itemId,
 59:             @AuthenticationPrincipal UserPrincipal currentUser) {
 60:
```

```
61:         itemService.deleteItem(itemId, currentUser);
62:
63:         ApiResponse response = ApiResponse.builder()
64:                 .success(true)
65:                 .message("Item deleted successfully")
66:                 .build();
67:
68:         return ResponseEntity.ok(response);
69:     }
70: }
```

■ 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.security.UserPrincipal;
 6: import com.lostandfound.service.MessageService;
 7: import jakarta.validation.Valid;
 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: @RestController
14: @RequestMapping("/messages")
15: @RequiredArgsConstructor
16: public class MessageController {
17:
18:     private final MessageService messageService;
19:
20:     @PostMapping
21:     public ResponseEntity<ApiResponse> sendMessage(
22:             @Valid @RequestBody MessageRequest request,
23:             @AuthenticationPrincipal UserPrincipal currentUser) {
24:
25:         ApiResponse response = messageService.sendMessage(request, currentUser);
26:         return ResponseEntity.ok(response);
27:     }
28:
29:     @PostMapping("/reply")
30:     public ResponseEntity<ApiResponse> replyMessage(
31:             @Valid @RequestBody MessageRequest request,
32:             @AuthenticationPrincipal UserPrincipal currentUser) {
33:
34:         ApiResponse response = messageService.sendMessage(request, currentUser);
35:         return ResponseEntity.ok(response);
36:     }
37: }
```

■ 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:
```

```
        logger.warn("Bad request: {}", ex.getMessage());

        ApiResponse response = ApiResponse.builder()
            .success(false)
            .message(ex.getMessage())
            .build();
        return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
    }

    @ExceptionHandler(UnauthorizedException.class)
    public ResponseEntity<ApiResponse> handleUnauthorizedException(
            UnauthorizedException ex, WebRequest request) {

        logger.warn("Unauthorized access: {}", ex.getMessage());

        ApiResponse response = ApiResponse.builder()
            .success(false)
            .message(ex.getMessage())
            .build();
        return new ResponseEntity<>(response, HttpStatus.UNAUTHORIZED);
    }

    @ExceptionHandler(BadCredentialsException.class)
    public ResponseEntity<ApiResponse> handleBadCredentialsException(
            BadCredentialsException ex, WebRequest request) {

        logger.warn("Invalid credentials attempt");

        ApiResponse response = ApiResponse.builder()
            .success(false)
            .message("Invalid email or password")
            .build();
        return new ResponseEntity<>(response, HttpStatus.UNAUTHORIZED);
    }

    @ExceptionHandler(UsernameNotFoundException.class)
    public ResponseEntity<ApiResponse> handleUsernameNotFoundException(
            UsernameNotFoundException ex, WebRequest request) {

        logger.warn("User not found");

        // Don't reveal if user exists or not for security
        ApiResponse response = ApiResponse.builder()
            .success(false)
            .message("Invalid email or password")
            .build();
        return new ResponseEntity<>(response, HttpStatus.UNAUTHORIZED);
    }

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity<Map<String, Object>> handleValidationExceptions(
            MethodArgumentNotValidException ex) {

        Map<String, String> errors = new HashMap<>();
        ex.getBindingResult().getAllErrors().forEach((error) -> {
            String fieldName = ((FieldError) error).getField();
            String errorMessage = error.getDefaultMessage();
            errors.put(fieldName, errorMessage);
        });

        logger.warn("Validation errors: {}", errors);

        Map<String, Object> response = new HashMap<>();
        response.put("success", false);
        response.put("message", "Validation failed");
        response.put("errors", errors);

        return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
    }

    @ExceptionHandler(MaxUploadSizeExceededException.class)
    public ResponseEntity<ApiResponse> handleMaxUploadSizeExceededException(
            MaxUploadSizeExceededException ex) {

        logger.warn("File size exceeded");

        ApiResponse response = ApiResponse.builder()
            .success(false)
            .message("File size exceeds maximum allowed size of 5MB")
            .build();
        return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
    }

    @ExceptionHandler(Exception.class)
    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 lombok.AllArgsConstructor;
 19: import lombok.Data;
 20: import lombok.NoArgsConstructor;
 21:
 22: @Entity
 23: @Table(name = "items")
 24: @Data
 25: @NoArgsConstructor
 26: @AllArgsConstructor
 27: public class Item {
 28:
 29:     @Id
 30:     @GeneratedValue(strategy = GenerationType.IDENTITY)
 31:     private Long id;
 32:
 33:     @Column(nullable = false)
 34:     private String name;
 35:
 36:     @Column(nullable = false, columnDefinition = "TEXT")
 37:     private String description;
 38:
 39:     @Column(nullable = false)
 40:     private String location;
 41:
 42:     @Enumerated(EnumType.STRING)
 43:     @Column(nullable = false)
 44:     private Status status = Status.LOST;
 45:
 46:     @Column(name = "image")
 47:     private String image;
 48:
 49:     @ManyToOne(fetch = FetchType.LAZY)
 50:     @JoinColumn(name = "created_by", nullable = false)
 51:     private User createdBy;
 52:
 53:     @CreationTimestamp
 54:     @Column(name = "created_at", nullable = false, updatable = false)
 55:     private LocalDateTime createdAt;
 56:
 57:     public enum Status {
 58:         LOST, FOUND, CLAIMED
 59:     }
 60: }
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\model\Message.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 = "messages")
 22: @Data
 23: @NoArgsConstructor
 24: @AllArgsConstructor
 25: public class Message {
 26:
 27:     @Id
 28:     @GeneratedValue(strategy = GenerationType.IDENTITY)
 29:     private Long id;
 30:
 31:     @ManyToOne(fetch = FetchType.LAZY)
 32:     @JoinColumn(name = "sender_id", nullable = false)
 33:     private User sender;
```

```
 34:
 35:     @ManyToOne(fetch = FetchType.LAZY)
 36:     @JoinColumn(name = "receiver_id", nullable = false)
 37:     private User receiver;
 38:
 39:     @ManyToOne(fetch = FetchType.LAZY)
 40:     @JoinColumn(name = "item_id", nullable = false)
 41:     private Item item;
 42:
 43:     @Column(nullable = false, columnDefinition = "TEXT")
 44:     private String message;
 45:
 46:     @CreationTimestamp
 47:     @Column(name = "sent_at", nullable = false, updatable = false)
 48:     private LocalDateTime sentAt;
 49: }
```

■ 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> findAllByOrderBySentAtDesc();
15: }
```

■ 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:     void deleteByUser(User user);
 24:
 25:     @Modifying
 26:     @Query("DELETE FROM RefreshToken rt WHERE rt.expiryDate < ?1")
 27:     void 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:
```

```
        return UserPrincipal.create(user);
    }
}
```

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

```
package com.lostandfound.security;

import com.lostandfound.util.CookieUtil;
import jakarta.servlet.FilterChain;
import jakarta.servlet.ServletException;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.stereotype.Component;
import org.springframework.util.StringUtils;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

@Component
@RequiredArgsConstructor
public class JwtAuthenticationFilter extends OncePerRequestFilter {

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

    private final JwtTokenProvider tokenProvider;
    private final CustomUserDetailsService customUserDetailsService;
    private final CookieUtil cookieUtil;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {
        try {
            // Try to get JWT from cookie first, then fall back to header
            String jwt = getJwtFromCookie(request);
            if (!StringUtils.hasText(jwt)) {
                jwt = getJwtFromHeader(request);
            }

            if (StringUtils.hasText(jwt) && tokenProvider.validateToken(jwt)) {
                Long userId = tokenProvider.getUserIdFromToken(jwt);
                UserDetails userDetails = customUserDetailsService.loadUserById(userId);

                UsernamePasswordAuthenticationToken authentication =
                        new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
                authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

                SecurityContextHolder.getContext().setAuthentication(authentication);
                logger.debug("Set authentication for user: {}", userId);
            }
        } catch (Exception ex) {
            logger.error("Could not set user authentication in security context", ex);
        }

        filterChain.doFilter(request, response);
    }

    /**
     * Get JWT from cookie
     */
    private String getJwtFromCookie(HttpServletRequest request) {
        return cookieUtil.getAccessToken(request).orElse(null);
    }

    /**
     * Get JWT from Authorization header (fallback for API clients)
     */
    private String getJwtFromHeader(HttpServletRequest request) {
        String bearerToken = request.getHeader("Authorization");
        if (StringUtils.hasText(bearerToken) && bearerToken.startsWith("Bearer ")) {
            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
106:      */
107:     private String getClientIP(HttpServletRequest request) {
108:         String xfHeader = request.getHeader("X-Forwarded-For");
109:         if (xfHeader == null || xfHeader.isEmpty()) {
110:             return request.getRemoteAddr();
111:         }
112:         // X-Forwarded-For can contain multiple IPs, get the first one
113:         return xfHeader.split(",")[0].trim();
114:     }
115:
116:     /**
117:      * Skip rate limiting for health check endpoints
118:      */
119:     @Override
120:     protected boolean shouldNotFilter(HttpServletRequest request) {
121:         String path = request.getRequestURI();
122:         return path.startsWith("/actuator/health") ||
123:                path.startsWith("/actuator/info");
124:     }
125: }
```

■ 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.model.Claim;
  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.UserRepository;
 13: import com.lostandfound.security.UserPrincipal;
 14: import lombok.RequiredArgsConstructor;
 15: import org.springframework.stereotype.Service;
 16: import org.springframework.transaction.annotation.Transactional;
 17:
 18: import java.util.List;
 19: import java.util.stream.Collectors;
 20:
 21: @Service
 22: @RequiredArgsConstructor
 23: public class ClaimService {
 24:
 25:     private final ClaimRepository claimRepository;
 26:     private final ItemRepository itemRepository;
 27:     private final UserRepository userRepository;
 28:
 29:     @Transactional
 30:     public ApiResponse claimItem(Long itemId, UserPrincipal currentUser) {
 31:         Item item = itemRepository.findById(itemId)
 32:                 .orElseThrow(() -> new ResourceNotFoundException("Item", "id", itemId));
 33:
 34:         User user = userRepository.findById(currentUser.getId())
 35:                 .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
    | .getId()));
 36:
 37:         if (item.getStatus() == Item.Status.CLAIMED) {
 38:             throw new BadRequestException("Item already claimed");
 39:         }
 40:
 41:         if (claimRepository.existsByItemAndClaimedBy(item, user)) {
 42:             throw new BadRequestException("You have already claimed this item");
 43:         }
 44:
 45:         Claim claim = new Claim();
 46:         claim.setItem(item);
 47:         claim.setClaimedBy(user);
 48:         claim.setClaimantName(user.getName());
 49:         claim.setClaimantEmail(user.getEmail());
 50:
 51:         claimRepository.save(claim);
 52:
 53:         // Update item status
 54:         item.setStatus(Item.Status.CLAIMED);
 55:         itemRepository.save(item);
 56:
 57:         return ApiResponse.builder()
```

```
 58:                 .success(true)
 59:                 .message("Item claimed successfully")
 60:                 .build();
 61:     }
 62:
 63:     @Transactional(readOnly = true)
 64:     public List<ClaimResponse> getUserClaims(UserPrincipal currentUser) {
 65:         User user = userRepository.findById(currentUser.getId())
 66:                 .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
    | .getId()));
 67:
 68:         List<Claim> claims = claimRepository.findByClaimedByOrderByClaimedAtDesc(user);
 69:         return claims.stream()
 70:                 .map(this::mapToClaimResponse)
 71:                 .collect(Collectors.toList());
 72:     }
 73:
 74:     @Transactional
 75:     public void deleteClaim(Long claimId) {
 76:         Claim claim = claimRepository.findById(claimId)
 77:                 .orElseThrow(() -> new ResourceNotFoundException("Claim", "id", claimId));
 78:
 79:         Item item = claim.getItem();
 80:
 81:         claimRepository.delete(claim);
 82:
 83:         // Update item status back to FOUND if it was CLAIMED
 84:         if (item.getStatus() == Item.Status.CLAIMED) {
 85:             item.setStatus(Item.Status.FOUND);
 86:             itemRepository.save(item);
 87:         }
 88:     }
 89:
 90:     private ClaimResponse mapToClaimResponse(Claim claim) {
 91:         return ClaimResponse.builder()
 92:                 .id(claim.getId())
 93:                 .itemId(claim.getItem().getId())
 94:                 .itemName(claim.getItem().getName())
 95:                 .description(claim.getItem().getDescription())
 96:                 .location(claim.getItem().getLocation())
 97:                 .claimedBy(claim.getClaimedBy().getId())
 98:                 .claimerName(claim.getClaimedBy().getName())
 99:                 .claimantName(claim.getClaimantName())
100:                 .claimantEmail(claim.getClaimantEmail())
101:                 .claimedAt(claim.getClaimedAt())
102:                 .build();
103:     }
104: }
```

■ 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.springframework.stereotype.Service;
  7: import org.springframework.util.StringUtils;
  8: import org.springframework.web.multipart.MultipartFile;
  9:
 10: import java.io.IOException;
 11: import java.nio.file.Files;
 12: import java.nio.file.Path;
 13: import java.nio.file.Paths;
 14: import java.nio.file.StandardCopyOption;
 15: import java.util.UUID;
 16:
 17: @Service
 18: @RequiredArgsConstructor
 19: public class FileStorageService {
 20:
 21:     private final FileStorageProperties fileStorageProperties;
 22:
 23:     public String storeFile(MultipartFile file) {
 24:         // Normalize file name
 25:         String originalFileName = StringUtils.cleanPath(file.getOriginalFilename());
 26:
 27:         try {
 28:             // Check if the file's name contains invalid characters
 29:             if (originalFileName.contains("..")) {
 30:                 throw new BadRequestException("Filename contains invalid path sequence " +
   |  originalFileName);
 31:             }
 32:
 33:             // Create unique filename
 34:             String fileExtension = "";
 35:             if (originalFileName.contains(".")) {
 36:                 fileExtension = originalFileName.substring(originalFileName.lastIndexOf(".
   | "));
 37:             }
 38:             String uniqueFileName = UUID.randomUUID().toString() + "_" + originalFileName;
 39:
```

```
            // Create directory if it doesn't exist
            Path fileStorageLocation = Paths.get(fileStorageProperties.getUploadDir())
                    .toAbsolutePath().normalize();
            Files.createDirectories(fileStorageLocation);

            // Copy file to the target location
            Path targetLocation = fileStorageLocation.resolve(uniqueFileName);
            Files.copy(file.getInputStream(), targetLocation, StandardCopyOption.REPLACE_E
   | XISTING);

            return "uploads/" + uniqueFileName;
        } catch (IOException ex) {
            throw new BadRequestException("Could not store file " + originalFileName + ".
   | Please try again!");
        }
    }

    public void deleteFile(String filePath) {
        try {
            if (filePath != null && filePath.startsWith("uploads/")) {
                String fileName = filePath.replace("uploads/", "");
                Path fileStorageLocation = Paths.get(fileStorageProperties.getUploadDir())
                        .toAbsolutePath().normalize();
                Path targetLocation = fileStorageLocation.resolve(fileName);
                Files.deleteIfExists(targetLocation);
            }
        } catch (IOException ex) {
            // Log error but don't throw exception
            System.err.println("Could not delete file: " + filePath);
        }
    }
}
```

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

```
package com.lostandfound.service;

import com.lostandfound.dto.request.ItemRequest;
import com.lostandfound.dto.response.ItemResponse;
import com.lostandfound.exception.BadRequestException;
import com.lostandfound.exception.ResourceNotFoundException;
import com.lostandfound.model.Item;
import com.lostandfound.model.User;
import com.lostandfound.repository.ClaimRepository;
import com.lostandfound.repository.ItemRepository;
import com.lostandfound.repository.MessageRepository;
import com.lostandfound.repository.UserRepository;
import com.lostandfound.security.UserPrincipal;
import lombok.RequiredArgsConstructor;
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 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) {
        User user = userRepository.findById(currentUser.getId())
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
   | .getId()));

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

        Item item = new Item();
        item.setName(request.getName());
        item.setDescription(request.getDescription());
        item.setLocation(request.getLocation());
        item.setStatus(Item.Status.valueOf(request.getStatus().toUpperCase()));
```

```
        item.setImage(imagePath);
        item.setCreatedBy(user);

        item = itemRepository.save(item);

        return mapToItemResponse(item);
    }

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

        List<Item> items = itemRepository.searchItems(search, itemStatus);
        return items.stream()
                .map(this::mapToItemResponse)
                .collect(Collectors.toList());
    }

    @Transactional(readOnly = true)
    public List<ItemResponse> getUserItems(UserPrincipal currentUser) {
        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 void deleteItem(Long itemId, UserPrincipal currentUser) {
        Item item = itemRepository.findById(itemId)
                .orElseThrow(() -> new ResourceNotFoundException("Item", "id", itemId));

        // Check if user is admin or item owner
        User user = userRepository.findById(currentUser.getId())
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
   | .getId()));

        if (!user.getRole().equals(User.Role.ADMIN) && !item.getCreatedBy().getId().equals
   | (currentUser.getId())) {
            throw new BadRequestException("You don't have permission to delete this item")
   | ;
        }

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

    private ItemResponse mapToItemResponse(Item item) {
        return ItemResponse.builder()
                .id(item.getId())
                .name(item.getName())
                .description(item.getDescription())
                .location(item.getLocation())
                .status(item.getStatus().name())
                .image(item.getImage())
                .createdBy(item.getCreatedBy().getId())
                .creatorName(item.getCreatedBy().getName())
                .createdAt(item.getCreatedAt())
                .build();
    }
}
```

■ 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.ResourceNotFoundException;
  7: import com.lostandfound.model.Item;
  8: import com.lostandfound.model.Message;
  9: import com.lostandfound.model.User;
 10: import com.lostandfound.repository.ItemRepository;
 11: import com.lostandfound.repository.MessageRepository;
 12: import com.lostandfound.repository.UserRepository;
 13: import com.lostandfound.security.UserPrincipal;
 14: import lombok.RequiredArgsConstructor;
 15: import org.springframework.stereotype.Service;
 16: import org.springframework.transaction.annotation.Transactional;
 17:
 18: import java.util.List;
 19: import java.util.stream.Collectors;
 20:
 21: @Service
 22: @RequiredArgsConstructor
 23: public class MessageService {
 24:
 25:     private final MessageRepository messageRepository;
 26:     private final UserRepository userRepository;
 27:     private final ItemRepository itemRepository;
 28:
 29:     @Transactional
 30:     public ApiResponse sendMessage(MessageRequest request, UserPrincipal currentUser) {
 31:         User sender = userRepository.findById(currentUser.getId())
 32:                 .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
    | .getId()));
 33:
 34:         User receiver = userRepository.findById(request.getReceiverId())
 35:                 .orElseThrow(() -> new ResourceNotFoundException("User", "id", request.get
    | ReceiverId()));
 36:
 37:         Item item = itemRepository.findById(request.getItemId())
 38:                 .orElseThrow(() -> new ResourceNotFoundException("Item", "id", request.get
    | ItemId()));
 39:
 40:         Message message = new Message();
 41:         message.setSender(sender);
 42:         message.setReceiver(receiver);
 43:         message.setItem(item);
 44:         message.setMessage(request.getMessage());
 45:
 46:         messageRepository.save(message);
 47:
 48:         return ApiResponse.builder()
 49:                 .success(true)
 50:                 .message("Message sent successfully")
 51:                 .build();
 52:     }
 53:
 54:     @Transactional(readOnly = true)
 55:     public List<MessageResponse> getUserMessages(UserPrincipal currentUser) {
 56:         User user = userRepository.findById(currentUser.getId())
 57:                 .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
    | .getId()));
 58:
 59:         List<Message> messages = messageRepository.findByReceiverOrderBySentAtDesc(user);
 60:         return messages.stream()
 61:                 .map(this::mapToMessageResponse)
 62:                 .collect(Collectors.toList());
 63:     }
 64:
 65:     private MessageResponse mapToMessageResponse(Message message) {
 66:         return MessageResponse.builder()
 67:                 .id(message.getId())
 68:                 .senderId(message.getSender().getId())
 69:                 .senderName(message.getSender().getName())
 70:                 .receiverId(message.getReceiver().getId())
 71:                 .itemId(message.getItem().getId())
 72:                 .itemName(message.getItem().getName())
 73:                 .message(message.getMessage())
 74:                 .sentAt(message.getSentAt())
 75:                 .build();
 76:     }
 77: }
```

■ 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.model.RefreshToken;
  5: import com.lostandfound.model.User;
  6: import com.lostandfound.repository.RefreshTokenRepository;
  7: import com.lostandfound.repository.UserRepository;
  8: import lombok.RequiredArgsConstructor;
  9: import org.springframework.beans.factory.annotation.Value;
 10: import org.springframework.stereotype.Service;
 11: import org.springframework.transaction.annotation.Transactional;
 12:
 13: import java.time.Instant;
 14: import java.util.Optional;
 15: import java.util.UUID;
 16:
 17: @Service
 18: @RequiredArgsConstructor
 19: public class RefreshTokenService {
 20:
 21:     private final RefreshTokenRepository refreshTokenRepository;
 22:     private final UserRepository userRepository;
 23:
 24:     @Value("${jwt.refresh.expiration}")
 25:     private Long refreshTokenDurationMs;
 26:
 27:     @Transactional
 28:     public RefreshToken createRefreshToken(Long userId, String ipAddress, String userAgent
    | ) {
 29:         User user = userRepository.findById(userId)
 30:                 .orElseThrow(() -> new BadRequestException("User not found"));
 31:
 32:         // Delete old refresh tokens for this user
 33:         refreshTokenRepository.deleteByUser(user);
 34:
 35:         RefreshToken refreshToken = RefreshToken.builder()
 36:                 .user(user)
 37:                 .token(UUID.randomUUID().toString())
 38:                 .expiryDate(Instant.now().plusMillis(refreshTokenDurationMs))
 39:                 .revoked(false)
 40:                 .ipAddress(ipAddress)
 41:                 .userAgent(userAgent)
 42:                 .build();
 43:
 44:         return refreshTokenRepository.save(refreshToken);
 45:     }
 46:
 47:     @Transactional(readOnly = true)
 48:     public Optional<RefreshToken> findByToken(String token) {
 49:         return refreshTokenRepository.findByToken(token);
 50:     }
 51:
 52:     @Transactional
 53:     public RefreshToken verifyExpiration(RefreshToken token) {
 54:         if (token.getExpiryDate().compareTo(Instant.now()) < 0) {
 55:             refreshTokenRepository.delete(token);
 56:             throw new BadRequestException("Refresh token expired. Please login again.");
 57:         }
 58:
 59:         if (token.isRevoked()) {
 60:             throw new BadRequestException("Refresh token has been revoked. Please login ag
    | ain.");
 61:         }
 62:
 63:         return token;
 64:     }
 65:
 66:     @Transactional
 67:     public void revokeToken(String token) {
 68:         refreshTokenRepository.findByToken(token).ifPresent(rt -> {
 69:             rt.setRevoked(true);
 70:             refreshTokenRepository.save(rt);
 71:         });
 72:     }
 73:
 74:     @Transactional
 75:     public void revokeAllUserTokens(User user) {
 76:         refreshTokenRepository.deleteByUser(user);
 77:     }
 78:
 79:     @Transactional
 80:     public void deleteExpiredTokens() {
 81:         refreshTokenRepository.deleteAllExpiredTokens(Instant.now());
```

```
 82:     }
 83: }
```

■ 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.model.RefreshToken;
  9: import com.lostandfound.model.User;
 10: import com.lostandfound.repository.UserRepository;
 11: import com.lostandfound.security.JwtTokenProvider;
 12: import com.lostandfound.security.UserPrincipal;
 13: import lombok.RequiredArgsConstructor;
 14: import org.slf4j.Logger;
 15: import org.slf4j.LoggerFactory;
 16: import org.springframework.beans.factory.annotation.Value;
 17: import org.springframework.security.authentication.AuthenticationManager;
 18: import org.springframework.security.authentication.BadCredentialsException;
 19: import org.springframework.security.authentication.LockedException;
 20: import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
 21: import org.springframework.security.core.Authentication;
 22: import org.springframework.security.core.context.SecurityContextHolder;
 23: import org.springframework.security.crypto.password.PasswordEncoder;
 24: import org.springframework.stereotype.Service;
 25: import org.springframework.transaction.annotation.Transactional;
 26:
 27: @Service
 28: @RequiredArgsConstructor
 29: public class UserService {
 30:
 31:     private static final Logger logger = LoggerFactory.getLogger(UserService.class);
 32:
 33:     private final UserRepository userRepository;
 34:     private final PasswordEncoder passwordEncoder;
 35:     private final AuthenticationManager authenticationManager;
 36:     private final JwtTokenProvider tokenProvider;
 37:     private final RefreshTokenService refreshTokenService;
 38:
 39:     @Value("${jwt.expiration}")
 40:     private Long jwtExpirationMs;
 41:
 42:     @Transactional
 43:     public AuthResponse registerUser(RegisterRequest request, String ipAddress, String use
    | rAgent) {
 44:         // Normalize email to lowercase
 45:         String email = request.getEmail().toLowerCase().trim();
 46:
 47:         // Check if email already exists
 48:         if (userRepository.existsByEmail(email)) {
 49:             logger.warn("Registration attempt with existing email: {}", email);
 50:             throw new BadRequestException("Email address is already registered");
 51:         }
 52:
 53:         // Validate password confirmation
 54:         if (!request.getPassword().equals(request.getConfirmPassword())) {
 55:             throw new BadRequestException("Passwords do not match");
 56:         }
 57:
 58:         // Create new user
 59:         User user = new User();
 60:         user.setName(request.getName().trim());
 61:         user.setEmail(email);
 62:         user.setPassword(passwordEncoder.encode(request.getPassword()));
 63:         user.setRole(User.Role.USER);
 64:
 65:         try {
 66:             user = userRepository.save(user);
 67:             logger.info("New user registered successfully: {}", email);
 68:         } catch (Exception e) {
 69:             logger.error("Error during user registration: {}", email, e);
 70:             throw new BadRequestException("Registration failed. Please try again.");
 71:         }
 72:
 73:         // Auto-login after registration
 74:         Authentication authentication = authenticationManager.authenticate(
 75:                 new UsernamePasswordAuthenticationToken(email, request.getPassword())
 76:         );
 77:
```

```
        SecurityContextHolder.getContext().setAuthentication(authentication);

        // Generate access token
        String accessToken = tokenProvider.generateToken(authentication);

        // Generate refresh token
        RefreshToken refreshToken = refreshTokenService.createRefreshToken(
                user.getId(), ipAddress, userAgent
        );

        // Build user DTO
        AuthResponse.UserDTO userDTO = AuthResponse.UserDTO.builder()
                .id(user.getId())
                .name(user.getName())
                .email(user.getEmail())
                .role(user.getRole().name())
                .build();

        return AuthResponse.builder()
                .success(true)
                .message("Registration successful")
                .accessToken(accessToken)
                .refreshToken(refreshToken.getToken())
                .tokenType("Bearer")
                .expiresIn(jwtExpirationMs / 1000) // Convert to seconds
                .user(userDTO)
                .build();
    }

    @Transactional
    public AuthResponse loginUser(LoginRequest request, String ipAddress, String userAgent
    | ) {
        String email = request.getEmail().toLowerCase().trim();

        try {
            // Authenticate user
            Authentication authentication = authenticationManager.authenticate(
                    new UsernamePasswordAuthenticationToken(email, request.getPassword())
            );

            SecurityContextHolder.getContext().setAuthentication(authentication);

            // Generate access token
            String accessToken = tokenProvider.generateToken(authentication);

            // Fetch user details
            User user = userRepository.findByEmail(email)
                    .orElseThrow(() -> new BadRequestException("User not found"));

            // Generate refresh token
            RefreshToken refreshToken = refreshTokenService.createRefreshToken(
                    user.getId(), ipAddress, userAgent
            );

            logger.info("User logged in successfully: {}", email);

            // Build user DTO
            AuthResponse.UserDTO userDTO = AuthResponse.UserDTO.builder()
                    .id(user.getId())
                    .name(user.getName())
                    .email(user.getEmail())
                    .role(user.getRole().name())
                    .build();

            return AuthResponse.builder()
                    .success(true)
                    .message("Login successful")
                    .accessToken(accessToken)
                    .refreshToken(refreshToken.getToken())
                    .tokenType("Bearer")
                    .expiresIn(jwtExpirationMs / 1000) // Convert to seconds
                    .user(userDTO)
                    .build();

        } catch (BadCredentialsException e) {
            logger.warn("Failed login attempt for email: {}", email);
            throw new BadCredentialsException("Invalid email or password");
        } catch (LockedException e) {
            logger.warn("Login attempt for locked account: {}", email);
            throw new BadRequestException("Account is locked. Please contact support.");
        } catch (Exception e) {
            logger.error("Error during login for email: {}", email, e);
            throw new BadRequestException("Login failed. Please try again.");
        }
    }
```

```
162:
163:     @Transactional
164:     public TokenRefreshResponse refreshToken(String refreshTokenStr) {
165:         return refreshTokenService.findByToken(refreshTokenStr)
166:                 .map(refreshTokenService::verifyExpiration)
167:                 .map(RefreshToken::getUser)
168:                 .map(user -> {
169:                     UserPrincipal userPrincipal = UserPrincipal.create(user);
170:                     Authentication auth = new UsernamePasswordAuthenticationToken(
171:                             userPrincipal, null, userPrincipal.getAuthorities()
172:                     );
173:
174:                     String newAccessToken = tokenProvider.generateToken(auth);
175:
176:                     return TokenRefreshResponse.builder()
177:                             .success(true)
178:                             .message("Token refreshed successfully")
179:                             .accessToken(newAccessToken)
180:                             .refreshToken(refreshTokenStr)
181:                             .tokenType("Bearer")
182:                             .build();
183:                 })
184:                 .orElseThrow(() -> new BadRequestException("Invalid refresh token"));
185:     }
186:
187:     @Transactional(readOnly = true)
188:     public User getUserById(Long userId) {
189:         return userRepository.findById(userId)
190:                 .orElseThrow(() -> new BadRequestException("User not found"));
191:     }
192:
193:     @Transactional(readOnly = true)
194:     public User getUserByEmail(String email) {
195:         return userRepository.findByEmail(email.toLowerCase().trim())
196:                 .orElseThrow(() -> new BadRequestException("User not found"));
197:     }
198: }
```

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

```
  1: package com.lostandfound.util;
  2:
  3: import jakarta.servlet.http.Cookie;
  4: import jakarta.servlet.http.HttpServletRequest;
  5: import jakarta.servlet.http.HttpServletResponse;
  6: import org.springframework.beans.factory.annotation.Value;
  7: import org.springframework.stereotype.Component;
  8:
  9: import java.util.Arrays;
 10: import java.util.Optional;
 11:
 12: @Component
 13: public class CookieUtil {
 14:
 15:     @Value("${cookie.domain:localhost}")
 16:     private String cookieDomain;
 17:
 18:     @Value("${cookie.secure:false}")
 19:     private boolean cookieSecure;
 20:
 21:     @Value("${cookie.same-site:Lax}")
 22:     private String cookieSameSite;
 23:
 24:     // Cookie names
 25:     public static final String ACCESS_TOKEN_COOKIE = "accessToken";
 26:     public static final String REFRESH_TOKEN_COOKIE = "refreshToken";
 27:
 28:     // Cookie max ages (in seconds)
 29:     private static final int ACCESS_TOKEN_MAX_AGE = 15 * 60; // 15 minutes
 30:     private static final int REFRESH_TOKEN_MAX_AGE = 7 * 24 * 60 * 60; // 7 days
 31:
 32:     /**
 33:      * Add access token cookie
 34:      */
 35:     public void addAccessTokenCookie(HttpServletResponse response, String token) {
 36:         addCookie(response, ACCESS_TOKEN_COOKIE, token, ACCESS_TOKEN_MAX_AGE, true);
 37:     }
 38:
 39:     /**
 40:      * Add refresh token cookie
 41:      */
 42:     public void addRefreshTokenCookie(HttpServletResponse response, String token) {
 43:         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) {
        return getCookie(request, ACCESS_TOKEN_COOKIE);
    }

    /**
     * Get refresh token from cookie
     */
    public Optional<String> getRefreshToken(HttpServletRequest request) {
        return getCookie(request, REFRESH_TOKEN_COOKIE);
    }

    /**
     * Delete cookie
     */
    public void deleteCookie(HttpServletResponse response, String name) {
        Cookie cookie = new Cookie(name, null);
        cookie.setPath("/");
        cookie.setHttpOnly(true);
        cookie.setMaxAge(0);
        cookie.setSecure(cookieSecure);

        response.addCookie(cookie);
    }

    /**
     * Delete access token cookie
     */
    public void deleteAccessToken(HttpServletResponse response) {
        deleteCookie(response, ACCESS_TOKEN_COOKIE);
    }

    /**
     * Delete refresh token cookie
     */
    public void deleteRefreshToken(HttpServletResponse response) {
        deleteCookie(response, REFRESH_TOKEN_COOKIE);
    }

    /**
     * Delete all auth cookies
     */
    public void deleteAllAuthCookies(HttpServletResponse response) {
```

```
        deleteAccessToken(response);
        deleteRefreshToken(response);
    }
}
```
