■ 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

```
==============================================================================================
package com.lostandfound;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableScheduling;

@SpringBootApplication
@EnableScheduling
public class LostAndFoundApplication {
    public static void main(String[] args) {
        SpringApplication.run(LostAndFoundApplication.class, args);
    }
}

----------------------------------------------------------------------------------------------
```

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

```
==============================================================================================
package com.lostandfound.annotation;

import com.lostandfound.config.RateLimitConfig;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * Custom annotation for method-level rate limiting
 * Usage: @RateLimit(type = RateLimitConfig.RateLimitType.AUTH)
 */
@Target({ElementType.METHOD, ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
public @interface RateLimit {
    RateLimitConfig.RateLimitType type() default RateLimitConfig.RateLimitType.API;
}

----------------------------------------------------------------------------------------------
```

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

```
==============================================================================================
package com.lostandfound.aspect;

import com.lostandfound.annotation.RateLimit;
import com.lostandfound.config.RateLimitConfig;
import com.lostandfound.exception.BadRequestException;
import io.github.bucket4j.Bucket;
import io.github.bucket4j.ConsumptionProbe;
import jakarta.servlet.http.HttpServletRequest;
import lombok.RequiredArgsConstructor;
import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.Around;
import org.aspectj.lang.annotation.Aspect;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Component;
import org.springframework.web.context.request.RequestContextHolder;
import org.springframework.web.context.request.ServletRequestAttributes;

/**
 * AOP Aspect for method-level rate limiting using @RateLimit annotation
 * This is optional and works alongside the filter-based rate limiting
 */
@Aspect
@Component
@RequiredArgsConstructor
public class RateLimitAspect {

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

    private final RateLimitConfig rateLimitConfig;

    @Around("@annotation(rateLimit)")
    public Object rateLimit(ProceedingJoinPoint joinPoint, RateLimit rateLimit) throws Thr
| owable {
        HttpServletRequest request = getCurrentRequest();

        if (request == null) {
            // If no request context, skip rate limiting
            return joinPoint.proceed();
        }

        String clientIp = getClientIP(request);
```

```java
        String method = joinPoint.getSignature().getName();

        RateLimitConfig.RateLimitType limitType = rateLimit.type();
        String bucketKey = clientIp + ":" + method + ":" + limitType.name();

        Bucket bucket = rateLimitConfig.resolveBucket(bucketKey, limitType);
        ConsumptionProbe probe = bucket.tryConsumeAndReturnRemaining(1);

        if (probe.isConsumed()) {
            logger.debug("Rate limit check passed for IP: {} on method: {}", clientIp, method);
            return joinPoint.proceed();
        } else {
            long waitForRefill = probe.getNanosToWaitForRefill() / 1_000_000_000;
            logger.warn("Rate limit exceeded for IP: {} on method: {}", clientIp, method);
            throw new BadRequestException(
                    "Rate limit exceeded. Please try again in " + waitForRefill + " seconds."
            );
        }
    }

    private HttpServletRequest getCurrentRequest() {
        ServletRequestAttributes attributes =
                (ServletRequestAttributes) RequestContextHolder.getRequestAttributes();
        return attributes != null ? attributes.getRequest() : null;
    }

    private String getClientIP(HttpServletRequest request) {
        String xfHeader = request.getHeader("X-Forwarded-For");
        String remoteAddr = request.getRemoteAddr();
        boolean isTrustedProxy = isTrustedProxy(remoteAddr);

        if (isTrustedProxy && xfHeader != null && !xfHeader.isEmpty()) {
            String clientIp = xfHeader.split(",")[0].trim();
            if (isValidIP(clientIp)) {
                return clientIp;
            }
        }

        return remoteAddr;
    }

    private boolean isTrustedProxy(String ip) {
        return "127.0.0.1".equals(ip) ||
                "0:0:0:0:0:0:0:1".equals(ip) ||
                "::1".equals(ip);
    }

    private boolean isValidIP(String ip) {
        if (ip == null || ip.isEmpty()) {
            return false;
        }
        String ipv4Pattern = "^((25[0-5]|(2[0-4]|1\\d|[1-9]|)\\d)\\.?\\b){4}$";
        String ipv6Pattern = "^(([0-9a-fA-F]{1,4}:){7}[0-9a-fA-F]{1,4}|([0-9a-fA-F]{1,4}:){1,7}:|([0-9a-fA-F]{1,4}:){1,6}:[0-9a-fA-F]{1,4}|([0-9a-fA-F]{1,4}:){1,5}(:[0-9a-fA-F]{1,4}){1,2})$";
        return ip.matches(ipv4Pattern) || ip.matches(ipv6Pattern);
    }
}
```

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

==

```java
package com.lostandfound.config;

import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.stereotype.Component;

import lombok.Data;

@Component
@ConfigurationProperties(prefix = "file")
@Data
public class FileStorageProperties {
    private String uploadDir;
}
```

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

==

```java
package com.lostandfound.config;

```

```java
import org.modelmapper.ModelMapper;
import org.modelmapper.convention.MatchingStrategies;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class ModelMapperConfig {

    @Bean
    public ModelMapper modelMapper() {
        ModelMapper modelMapper = new ModelMapper();
        modelMapper.getConfiguration()
                .setMatchingStrategy(MatchingStrategies.STRICT)
                .setSkipNullEnabled(true);
        return modelMapper;
    }
}
```

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

==

```java
package com.lostandfound.config;

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() {
        cache.clear();
    }

    /**
     * Get current cache size
     */
    public int getCacheSize() {
        return cache.size();
    }
}
```

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

===

```java
package com.lostandfound.config;

import com.lostandfound.security.JwtAuthenticationFilter;
import com.lostandfound.security.RateLimitFilter;
import lombok.RequiredArgsConstructor;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.annotation.authentication.configuration.Authent
| icationConfiguration;
import org.springframework.security.config.annotation.method.configuration.EnableMethodSec
| urity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigur
| er;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilte
| r;
import org.springframework.security.web.csrf.CookieCsrfTokenRepository;
import org.springframework.security.web.csrf.CsrfTokenRequestAttributeHandler;
import org.springframework.security.web.header.writers.ReferrerPolicyHeaderWriter;
import org.springframework.security.web.header.writers.StaticHeadersWriter;
import org.springframework.security.web.header.writers.XXssProtectionHeaderWriter;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;

import java.util.Arrays;
import java.util.List;

@Configuration
@EnableWebSecurity
@EnableMethodSecurity(prePostEnabled = true, securedEnabled = true)
@RequiredArgsConstructor
public class SecurityConfig {

    private final JwtAuthenticationFilter jwtAuthenticationFilter;
    private final RateLimitFilter rateLimitFilter;
    private final UserDetailsService userDetailsService;

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

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

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

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

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

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

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

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

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

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

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

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

                // Authentication Provider
                .authenticationProvider(authenticationProvider())

                // Add filters in correct order:
                // 1. First add JWT filter before UsernamePasswordAuthenticationFilter
                .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class)
                // 2. Then add Rate Limit filter before JWT filter
                .addFilterBefore(rateLimitFilter, JwtAuthenticationFilter.class)

                // Security Headers
                .headers(headers -> headers
                        // Content Security Policy
                        .contentSecurityPolicy(csp ->
                                csp.policyDirectives("default-src 'self'; " +
                                        "script-src 'self' 'unsafe-inline'; " +
                                        "style-src 'self' 'unsafe-inline'; " +
                                        "img-src 'self' data: https:; " +
                                        "font-src 'self' data:; " +
                                        "connect-src 'self'"))

                        // Frame Options - Prevent Clickjacking
                        .frameOptions(frame -> frame.deny())

                        // XSS Protection
                        .xssProtection(xss -> xss
                                .headerValue(XXssProtectionHeaderWriter.HeaderValue.ENABLED_MODE_BLOCK))

                        // HSTS - Force HTTPS
                        .httpStrictTransportSecurity(hsts -> hsts
                                .includeSubDomains(true)
                                .maxAgeInSeconds(31536000)) // 1 year

                        // Prevent MIME Sniffing
                        .contentTypeOptions(contentType -> {})

                        // Referrer Policy
                        .referrerPolicy(referrer ->
                                referrer.policy(ReferrerPolicyHeaderWriter.ReferrerPolicy.STRICT_ORIGIN_WHEN_CROSS_ORIGIN))

                        // Permissions Policy
                        .addHeaderWriter(new StaticHeadersWriter(
                                "Permissions-Policy",
                                "geolocation=(), microphone=(), camera=()"
                        ))
                );

        return http.build();
    }

    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();

        // Parse allowed origins from application.properties
        List<String> origins = Arrays.asList(allowedOrigins.split(","));
        configuration.setAllowedOriginPatterns(origins);

        // Allowed HTTP methods
        configuration.setAllowedMethods(Arrays.asList(
                "GET", "POST", "PUT", "DELETE", "PATCH", "OPTIONS"
        ));

        // Allowed headers
        configuration.setAllowedHeaders(Arrays.asList(
                "Authorization",
                "Content-Type",
                "X-Requested-With",
                "Accept",
                "Origin",
                "Access-Control-Request-Method",
                "Access-Control-Request-Headers",
                "X-CSRF-TOKEN"
        ));

        // Exposed headers (so frontend can read them)
        configuration.setExposedHeaders(Arrays.asList(
                "Authorization",
                "X-CSRF-TOKEN",
                "Access-Control-Allow-Origin",
                "Access-Control-Allow-Credentials"
        ));

        // Allow credentials (cookies, authorization headers)
        configuration.setAllowCredentials(true);

        // Cache preflight response for 1 hour
        configuration.setMaxAge(3600L);

        UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", configuration);

        return source;
    }
}
```

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

==

```java
package com.lostandfound.config;

import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.ResourceHandlerRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class WebConfig implements WebMvcConfigurer {

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

    @Value("${file.upload-dir}")
    private String uploadDir;

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
                .allowedOrigins(allowedOrigins.split(","))
                .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
                .allowedHeaders("*")
                .allowCredentials(true)
                .maxAge(3600);
    }

    @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

==

```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.BadRequestException;
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 com.lostandfound.service.UserService;
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
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
```

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

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

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

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

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

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

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

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

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

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

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

        return ResponseEntity.ok(response);
    }

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

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

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

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

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

        return ResponseEntity.ok(response);
    }

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

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

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

        claimService.deleteClaim(claimId);

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

        return ResponseEntity.ok(response);
    }

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

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

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

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

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

        return ResponseEntity.ok(response);
    }

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

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

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

        feedbackService.deleteFeedback(feedbackId);

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

        return ResponseEntity.ok(response);
    }

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

```java

        List<ItemResponse> items = itemService.searchItems(search, status);

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

        return ResponseEntity.ok(response);
    }

    @GetMapping("/claims")
    public ResponseEntity<Map<String, Object>> getAllClaims() {
        List<ClaimResponse> claims = claimService.getUserClaims(null);

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

        return ResponseEntity.ok(response);
    }

    @GetMapping("/users")
    public ResponseEntity<Map<String, Object>> getAllUsers() {
        List<User> users = userRepository.findAll();

        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("success", true);
        response.put("message", "Users retrieved successfully");
        response.put("users", userList);
        response.put("count", userList.size());

        return ResponseEntity.ok(response);
    }

    @GetMapping("/feedback")
    public ResponseEntity<Map<String, Object>> getAllFeedback() {
        List<FeedbackResponse> feedback = feedbackService.getAllFeedback();

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

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

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

==

```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.exception.BadRequestException;
import com.lostandfound.security.UserPrincipal;
import com.lostandfound.service.RefreshTokenService;
import com.lostandfound.service.UserService;
import com.lostandfound.util.CookieUtil;
import jakarta.servlet.http.HttpServletRequest;
import jakarta.servlet.http.HttpServletResponse;
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;
    private final CookieUtil cookieUtil; // Add this

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

        String ipAddress = getClientIP(httpRequest);
        String userAgent = httpRequest.getHeader("User-Agent");

        AuthResponse response = userService.registerUser(request, ipAddress, userAgent);

        // Set refresh token in httpOnly cookie
        cookieUtil.addRefreshTokenCookie(httpResponse, response.getRefreshToken());

        // Remove refresh token from response body (security best practice)
        response.setRefreshToken(null);

        return ResponseEntity.ok(response);
    }

    @PostMapping("/login")
    public ResponseEntity<AuthResponse> loginUser(
            @Valid @RequestBody LoginRequest request,
            HttpServletRequest httpRequest,
            HttpServletResponse httpResponse) { // Add response parameter

        String ipAddress = getClientIP(httpRequest);
        String userAgent = httpRequest.getHeader("User-Agent");

        AuthResponse response = userService.loginUser(request, ipAddress, userAgent);

        // Set refresh token in httpOnly cookie
        cookieUtil.addRefreshTokenCookie(httpResponse, response.getRefreshToken());

        // Remove refresh token from response body
        response.setRefreshToken(null);

        return ResponseEntity.ok(response);
    }

    @PostMapping("/refresh")
    public ResponseEntity<TokenRefreshResponse> refreshToken(
            HttpServletRequest httpRequest,
            HttpServletResponse httpResponse) {

        // Get refresh token from cookie instead of request body
        String refreshToken = cookieUtil.getRefreshToken(httpRequest)
                .orElseThrow(() -> new BadRequestException("Refresh token not found"));

        TokenRefreshResponse response = userService.refreshToken(refreshToken);

        // Update refresh token cookie
        cookieUtil.addRefreshTokenCookie(httpResponse, response.getRefreshToken());

        // Remove refresh token from response body
        response.setRefreshToken(null);

        return ResponseEntity.ok(response);
    }

    @PostMapping("/logout")
    public ResponseEntity<ApiResponse> logout(
            @AuthenticationPrincipal UserPrincipal currentUser,
            HttpServletRequest httpRequest,
            HttpServletResponse httpResponse) {

        // User must be authenticated to logout
        if (currentUser == null) {
            throw new BadRequestException("You are not logged in");
        }

        // Get refresh token from cookie
        String refreshToken = cookieUtil.getRefreshToken(httpRequest).orElse(null);

        if (refreshToken != null) {
            // Verify the token belongs to the current user before revoking
            refreshTokenService.revokeTokenForUser(refreshToken, currentUser.getId());
        } else {
            // Revoke all tokens for this user
            refreshTokenService.revokeAllUserTokens(currentUser.getId());
        }

        // Clear auth cookies
        cookieUtil.deleteAllAuthCookies(httpResponse);

        ApiResponse response = ApiResponse.builder()
                .success(true)
                .message("Logged out successfully")
                .build();

        return ResponseEntity.ok(response);
    }

    @GetMapping("/validate")
    public ResponseEntity<ApiResponse> validateToken(
            @AuthenticationPrincipal UserPrincipal currentUser) {

        if (currentUser == null) {
            throw new BadRequestException("Invalid or expired token");
        }

        ApiResponse response = ApiResponse.builder()
                .success(true)
                .message("Token is valid")
                .data(currentUser.getEmail())
                .build();

        return ResponseEntity.ok(response);
    }

    @GetMapping("/me")
    public ResponseEntity<ApiResponse> getCurrentUser(
            @AuthenticationPrincipal UserPrincipal currentUser) {

        if (currentUser == null) {
            throw new BadRequestException("Not authenticated");
        }

        AuthResponse.UserDTO userDTO = AuthResponse.UserDTO.builder()
                .id(currentUser.getId())
                .name(currentUser.getName())
                .email(currentUser.getEmail())
                .role(currentUser.getAuthorities().iterator().next().getAuthority().replace("ROLE_", ""))
                .build();

        ApiResponse response = ApiResponse.builder()
                .success(true)
                .message("User details retrieved successfully")
                .data(userDTO)
                .build();

        return ResponseEntity.ok(response);
    }

    private String getClientIP(HttpServletRequest request) {
        String xfHeader = request.getHeader("X-Forwarded-For");
        String remoteAddr = request.getRemoteAddr();
        boolean isTrustedProxy = isTrustedProxy(remoteAddr);

        if (isTrustedProxy && xfHeader != null && !xfHeader.isEmpty()) {
            String clientIp = xfHeader.split(",")[0].trim();
            if (isValidIP(clientIp)) {
                return clientIp;
            }
        }

        return remoteAddr;
    }

    private boolean isTrustedProxy(String ip) {
        return "127.0.0.1".equals(ip) ||
                "0:0:0:0:0:0:0:1".equals(ip) ||
                "::1".equals(ip);
    }

    private boolean isValidIP(String ip) {
        if (ip == null || ip.isEmpty()) {
            return false;
        }
        String ipv4Pattern = "^((25[0-5]|(2[0-4]|1\\d|[1-9]|)\\d)\\.?\\b){4}$";
        return ip.matches(ipv4Pattern);
    }
}
```

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

==

```
package com.lostandfound.controller;

import com.lostandfound.dto.response.ApiResponse;
import com.lostandfound.dto.response.ClaimResponse;
import com.lostandfound.exception.BadRequestException;
import com.lostandfound.security.UserPrincipal;
import com.lostandfound.service.ClaimService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.*;

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

@RestController
@RequestMapping("/claims")
@RequiredArgsConstructor
public class ClaimController {

    private final ClaimService claimService;

    @PostMapping("/item/{itemId}")
    public ResponseEntity<ApiResponse> claimItem(
            @PathVariable Long itemId,
            @AuthenticationPrincipal UserPrincipal currentUser) {

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

        ApiResponse response = claimService.claimItem(itemId, currentUser);
        return ResponseEntity.ok(response);
    }

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

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

        List<ClaimResponse> claims = claimService.getUserClaims(currentUser);

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

        return ResponseEntity.ok(response);
    }

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

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

        claimService.deleteClaimByUser(claimId, currentUser);

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

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

```java
}
```

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

===

```java
package com.lostandfound.controller;

import com.lostandfound.dto.response.ClaimResponse;
import com.lostandfound.dto.response.DashboardResponse;
import com.lostandfound.dto.response.ItemResponse;
import com.lostandfound.dto.response.MessageResponse;
import com.lostandfound.exception.BadRequestException;
import com.lostandfound.security.UserPrincipal;
import com.lostandfound.service.ClaimService;
import com.lostandfound.service.ItemService;
import com.lostandfound.service.MessageService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController
@RequestMapping("/dashboard")
@RequiredArgsConstructor
public class DashboardController {

    private final ItemService itemService;
    private final ClaimService claimService;
    private final MessageService messageService;

    @GetMapping
    public ResponseEntity<DashboardResponse> getDashboard(
            @AuthenticationPrincipal UserPrincipal currentUser) {

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

        List<ItemResponse> items = itemService.getUserItems(currentUser);
        List<ClaimResponse> claims = claimService.getUserClaims(currentUser);
        List<MessageResponse> messages = messageService.getUserMessages(currentUser);

        String role = currentUser.getAuthorities().iterator().next()
                .getAuthority().replace("ROLE_", "");

        DashboardResponse.UserInfo userInfo = DashboardResponse.UserInfo.builder()
                .name(currentUser.getName())
                .email(currentUser.getEmail())
                .role(role)
                .build();

        DashboardResponse response = DashboardResponse.builder()
                .user(userInfo)
                .items(items)
                .claims(claims)
                .messages(messages)
                .build();

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

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

===

```java
package com.lostandfound.controller;

import com.lostandfound.dto.request.FeedbackRequest;
import com.lostandfound.dto.response.ApiResponse;
import com.lostandfound.exception.BadRequestException;
import com.lostandfound.security.UserPrincipal;
import com.lostandfound.service.FeedbackService;
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.*;
```

```java

@RestController
@RequestMapping("/feedback")
@RequiredArgsConstructor
public class FeedbackController {

    private final FeedbackService feedbackService;

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

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

        // Additional validation for feedback content
        if (request.getFeedback() == null || request.getFeedback().trim().isEmpty()) {
            throw new BadRequestException("Feedback content cannot be empty");
        }

        if (request.getFeedback().trim().length() < 10) {
            throw new BadRequestException("Feedback must be at least 10 characters long");
        }

        if (request.getFeedback().trim().length() > 2000) {
            throw new BadRequestException("Feedback cannot exceed 2000 characters");
        }

        ApiResponse response = feedbackService.submitFeedback(request, currentUser);
        return ResponseEntity.ok(response);
    }
}
```

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

==

```java
package com.lostandfound.controller;

import com.lostandfound.dto.request.ItemRequest;
import com.lostandfound.dto.response.ApiResponse;
import com.lostandfound.dto.response.ItemResponse;
import com.lostandfound.exception.BadRequestException;
import com.lostandfound.security.UserPrincipal;
import com.lostandfound.service.ItemService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.multipart.MultipartFile;

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

@RestController
@RequestMapping("/items")
@RequiredArgsConstructor
public class ItemController {

    private final ItemService itemService;

    @PostMapping
    public ResponseEntity<ApiResponse> createItem(
            @Valid @ModelAttribute ItemRequest request,
            @RequestParam(value = "image", required = false) MultipartFile image,
            @AuthenticationPrincipal UserPrincipal currentUser) {

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

        // Validate image if provided
        if (image != null && !image.isEmpty()) {
            // Check file size (5MB max)
            if (image.getSize() > 5 * 1024 * 1024) {
                throw new BadRequestException("Image size must not exceed 5MB");
            }

            // Check file type
            String contentType = image.getContentType();
            if (contentType == null || !contentType.startsWith("image/")) {
                throw new BadRequestException("Only image files are allowed");
            }
        }

        ItemResponse itemResponse = itemService.createItem(request, image, currentUser);

        ApiResponse response = ApiResponse.builder()
                .success(true)
                .message("Item registered successfully")
                .data(itemResponse)
                .build();

        return ResponseEntity.ok(response);
    }

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

        List<ItemResponse> items = itemService.searchItems(search, status);

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

        return ResponseEntity.ok(response);
    }

    @GetMapping("/{itemId}")
    public ResponseEntity<ApiResponse> getItemById(@PathVariable Long itemId) {
        ItemResponse item = itemService.getItemById(itemId);

        ApiResponse response = ApiResponse.builder()
                .success(true)
                .message("Item retrieved successfully")
                .data(item)
                .build();

        return ResponseEntity.ok(response);
    }

    @PutMapping("/{itemId}")
    public ResponseEntity<ApiResponse> updateItem(
            @PathVariable Long itemId,
            @Valid @ModelAttribute ItemRequest request,
            @RequestParam(value = "image", required = false) MultipartFile image,
            @AuthenticationPrincipal UserPrincipal currentUser) {

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

        // Validate image if provided
        if (image != null && !image.isEmpty()) {
            if (image.getSize() > 5 * 1024 * 1024) {
                throw new BadRequestException("Image size must not exceed 5MB");
            }

            String contentType = image.getContentType();
            if (contentType == null || !contentType.startsWith("image/")) {
                throw new BadRequestException("Only image files are allowed");
            }
        }

        ItemResponse itemResponse = itemService.updateItem(itemId, request, image, current
| User);

        ApiResponse response = ApiResponse.builder()
                .success(true)
                .message("Item updated successfully")
                .data(itemResponse)
                .build();

        return ResponseEntity.ok(response);
    }

    @DeleteMapping("/{itemId}")
    public ResponseEntity<ApiResponse> deleteItem(
            @PathVariable Long itemId,
```

```java
            @AuthenticationPrincipal UserPrincipal currentUser) {

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

        itemService.deleteItem(itemId, currentUser);

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

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

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

==

```java
package com.lostandfound.controller;

import com.lostandfound.dto.request.MessageRequest;
import com.lostandfound.dto.response.ApiResponse;
import com.lostandfound.dto.response.MessageResponse;
import com.lostandfound.exception.BadRequestException;
import com.lostandfound.security.UserPrincipal;
import com.lostandfound.service.MessageService;
import jakarta.validation.Valid;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.*;

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

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

    private final MessageService messageService;

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

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

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

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

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

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

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

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

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

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

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

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

        return ResponseEntity.ok(response);
    }

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

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

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

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

        return ResponseEntity.ok(response);
    }

    @DeleteMapping("/{messageId}")
    public ResponseEntity<ApiResponse> deleteMessage(
            @PathVariable Long messageId,
            @AuthenticationPrincipal UserPrincipal currentUser) {

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

        messageService.deleteMessage(messageId, currentUser);

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

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

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

==

```java
package com.lostandfound.controller;

import com.lostandfound.config.RateLimitConfig;
import com.lostandfound.dto.response.ApiResponse;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;

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

@RestController
@RequestMapping("/admin/rate-limit")
@PreAuthorize("hasRole('ADMIN')")
@RequiredArgsConstructor
public class RateLimitController {

    private final RateLimitConfig rateLimitConfig;

    /**
     * Get rate limit statistics
     */
    @GetMapping("/stats")
    public ResponseEntity<ApiResponse> getRateLimitStats() {
        Map<String, Object> stats = new HashMap<>();
        stats.put("activeBuckets", rateLimitConfig.getCacheSize());
        stats.put("rateLimitTypes", getRateLimitInfo());

        ApiResponse response = ApiResponse.builder()
                .success(true)
                .message("Rate limit statistics")
                .data(stats)
                .build();

        return ResponseEntity.ok(response);
    }

    /**
     * Clear rate limit for specific IP
     */
    @DeleteMapping("/clear/{ip}")
    public ResponseEntity<ApiResponse> clearRateLimitForIp(@PathVariable String ip) {
        // Clear all buckets for this IP across all limit types
        for (RateLimitConfig.RateLimitType type : RateLimitConfig.RateLimitType.values())
 | {
            String key = ip + ":" + type.name();
            rateLimitConfig.clearBucket(key);
        }

        ApiResponse response = ApiResponse.builder()
                .success(true)
                .message("Rate limit cleared for IP: " + ip)
                .build();

        return ResponseEntity.ok(response);
    }

    /**
     * Clear all rate limits (use with caution)
     */
    @DeleteMapping("/clear-all")
    public ResponseEntity<ApiResponse> clearAllRateLimits() {
        rateLimitConfig.clearAllBuckets();

        ApiResponse response = ApiResponse.builder()
                .success(true)
                .message("All rate limits cleared")
                .build();

        return ResponseEntity.ok(response);
    }

    /**
     * Get rate limit configuration info
     */
    private Map<String, Object> getRateLimitInfo() {
        Map<String, Object> info = new HashMap<>();

        for (RateLimitConfig.RateLimitType type : RateLimitConfig.RateLimitType.values())
 | {
            Map<String, Object> typeInfo = new HashMap<>();
            typeInfo.put("capacity", type.getCapacity());
            typeInfo.put("refillDurationSeconds", type.getRefillDuration().getSeconds());
            info.put(type.name(), typeInfo);
        }

        return info;
    }
}
```

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

==

```java
package com.lostandfound.dto.request;

import jakarta.validation.constraints.NotBlank;
import lombok.Data;

@Data
public class FeedbackRequest {

    @NotBlank(message = "Feedback text is required")
    private String feedback;
}

-----------------------------------------------------------------------------------------------
```

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

```
===============================================================================================
package com.lostandfound.dto.request;

import jakarta.validation.constraints.NotBlank;
import lombok.Data;

@Data
public class ItemRequest {

    @NotBlank(message = "Item name is required")
    private String name;

    @NotBlank(message = "Description is required")
    private String description;

    @NotBlank(message = "Location is required")
    private String location;

    @NotBlank(message = "Status is required")
    private String status;
}

-----------------------------------------------------------------------------------------------
```

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

```
===============================================================================================
package com.lostandfound.dto.request;

import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Size;
import lombok.Data;

@Data
public class LoginRequest {

    @NotBlank(message = "Email is required")
    @Email(message = "Please provide a valid email address")
    @Size(max = 255, message = "Email must not exceed 255 characters")
    private String email;

    @NotBlank(message = "Password is required")
    @Size(min = 8, max = 128, message = "Password must be between 8 and 128 characters")
    private String password;
}

-----------------------------------------------------------------------------------------------
```

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

```
===============================================================================================
package com.lostandfound.dto.request;

import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;
import lombok.Data;

@Data
public class MessageRequest {

    @NotNull(message = "Receiver ID is required")
    private Long receiverId;

    @NotNull(message = "Item ID is required")
    private Long itemId;

    @NotBlank(message = "Message is required")
    private String message;
}

-----------------------------------------------------------------------------------------------
```

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

```
===============================================================================================
package com.lostandfound.dto.request;

import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.Pattern;
import jakarta.validation.constraints.Size;
import lombok.Data;
import lombok.AllArgsConstructor;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class RegisterRequest {

    @NotBlank(message = "Name is required")
    @Size(min = 2, max = 100, message = "Name must be between 2 and 100 characters")
    @Pattern(regexp = "^[a-zA-Z\\s]+$", message = "Name should only contain letters and spaces")
    private String name;

    @NotBlank(message = "Email is required")
    @Email(message = "Please provide a valid email address")
    @Size(max = 255, message = "Email must not exceed 255 characters")
    private String email;

    @NotBlank(message = "Password is required")
    @Size(min = 8, max = 128, message = "Password must be between 8 and 128 characters")
    @Pattern(
            regexp = "^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d)(?=.*[^a-zA-Z\\d]).{8,}$",
            message = "Password must contain at least one uppercase letter, one lowercase letter, one number, and one special character"
    )
    private String password;

    @NotBlank(message = "Confirm password is required")
    private String confirmPassword;
}

-----------------------------------------------------------------------------------------------
```

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

```
===============================================================================================
package com.lostandfound.dto.request;

import jakarta.validation.constraints.NotBlank;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
public class TokenRefreshRequest {

    @NotBlank(message = "Refresh token is required")
    private String refreshToken;
}

-----------------------------------------------------------------------------------------------
```

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

```
===============================================================================================
package com.lostandfound.dto.response;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class ApiResponse {
    private boolean success;
    private String message;
    private Object data;
}

-----------------------------------------------------------------------------------------------
```

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

```
===============================================================================================
package com.lostandfound.dto.response;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
```

```java
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class AuthResponse {
    private boolean success;
    private String message;
    private String accessToken;
    private String refreshToken;
    @Builder.Default
    private String tokenType = "Bearer";
    private Long expiresIn; // in seconds
    private UserDTO user;

    @Data
    @Builder
    @NoArgsConstructor
    @AllArgsConstructor
    public static class UserDTO {
        private Long id;
        private String name;
        private String email;
        private String role;
    }
}
```

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

==

```java
package com.lostandfound.dto.response;

import java.time.LocalDateTime;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class ClaimResponse {
    private Long id;
    private Long itemId;
    private String itemName;
    private String description;
    private String location;
    private Long claimedBy;
    private String claimerName;
    private String claimantName;
    private String claimantEmail;
    private LocalDateTime claimedAt;
}
```

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

==

```java
package com.lostandfound.dto.response;

import java.util.List;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class DashboardResponse {
    private UserInfo user;
    private List<ItemResponse> items;
    private List<ClaimResponse> claims;
    private List<MessageResponse> messages;

    @Data
    @Builder
    @NoArgsConstructor
    @AllArgsConstructor
    public static class UserInfo {
        private String name;
        private String email;
        private String role;
    }
}
```

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

==

```
package com.lostandfound.dto.response;

import java.time.LocalDateTime;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class FeedbackResponse {
    private Long id;
    private Long userId;
    private String userName;
    private String feedbackText;
    private LocalDateTime submittedAt;
}
```

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

==

```
package com.lostandfound.dto.response;

import java.time.LocalDateTime;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class ItemResponse {
    private Long id;
    private String name;
    private String description;
    private String location;
    private String status;
    private String image;
    private Long createdBy;
    private String creatorName;
    private LocalDateTime createdAt;
}
```

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

==

```
package com.lostandfound.dto.response;

import java.time.LocalDateTime;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class MessageResponse {
    private Long id;
    private Long senderId;
    private String senderName;
    private Long receiverId;
    private Long itemId;
    private String itemName;
    private String message;
    private LocalDateTime sentAt;
}
```

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

==

```java
package com.lostandfound.dto.response;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class TokenRefreshResponse {
    private boolean success;
    private String message;
    private String accessToken;
    private String refreshToken;
    @Builder.Default
    private String tokenType = "Bearer";
}
```

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

==

```java
package com.lostandfound.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(HttpStatus.BAD_REQUEST)
public class BadRequestException extends RuntimeException {

    public BadRequestException(String message) {
        super(message);
    }
}
```

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

==

```java
package com.lostandfound.exception;

import com.lostandfound.dto.response.ApiResponse;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;
import org.springframework.web.context.request.WebRequest;
import org.springframework.web.multipart.MaxUploadSizeExceededException;

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

@RestControllerAdvice
public class GlobalExceptionHandler {

    private static final Logger logger = LoggerFactory.getLogger(GlobalExceptionHandler.cl
  | ass);

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

        logger.warn("Resource not found: {}", ex.getMessage());

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

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

```

```java
        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(
            Exception ex, WebRequest request) {

        // Log the full exception for debugging
        logger.error("Unexpected error occurred", ex);

        // Don't expose internal error details to client
        ApiResponse response = ApiResponse.builder()
            .success(false)
            .message("An unexpected error occurred. Please try again later.")
            .build();
        return new ResponseEntity<>(response, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

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

```java
package com.lostandfound.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException {

    public ResourceNotFoundException(String message) {
        super(message);
    }

    public ResourceNotFoundException(String resourceName, String fieldName, Object fieldVa
| lue) {
        super(String.format("%s not found with %s: '%s'", resourceName, fieldName, fieldVa
| lue));
    }
}
```

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

```java
package com.lostandfound.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(HttpStatus.UNAUTHORIZED)
public class UnauthorizedException extends RuntimeException {

    public UnauthorizedException(String message) {
        super(message);
    }
}
```

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

```java
package com.lostandfound.model;

import java.time.LocalDateTime;

import org.hibernate.annotations.CreationTimestamp;

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

@Entity
@Table(name = "claims")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Claim {

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

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

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

    @Column(name = "claimant_name", nullable = false)
    private String claimantName;

    @Column(name = "claimant_email", nullable = false)
    private String claimantEmail;

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

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

==

```
package com.lostandfound.model;

import java.time.LocalDateTime;

import org.hibernate.annotations.CreationTimestamp;

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

@Entity
@Table(name = "feedback")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Feedback {

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

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

    @Column(name = "feedback_text", nullable = false, columnDefinition = "TEXT")
    private String feedbackText;

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

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

==

```
package com.lostandfound.model;

import java.time.LocalDateTime;

import org.hibernate.annotations.CreationTimestamp;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.EnumType;
import jakarta.persistence.Enumerated;
import jakarta.persistence.FetchType;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
```

```java
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.ManyToOne;
import jakarta.persistence.Table;
import jakarta.persistence.Version;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Table(name = "items")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Item {

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

    @Column(nullable = false)
    private String name;

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

    @Column(nullable = false)
    private String location;

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

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

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

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

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

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

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

==

```java
package com.lostandfound.model;

import java.time.LocalDateTime;

import org.hibernate.annotations.CreationTimestamp;

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

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

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

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

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

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

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

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

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

==

```java
package com.lostandfound.model;

import jakarta.persistence.*;
import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.hibernate.annotations.CreationTimestamp;

import java.time.Instant;
import java.time.LocalDateTime;

@Entity
@Table(name = "refresh_tokens")
@Data
@Builder
@NoArgsConstructor
@AllArgsConstructor
public class RefreshToken {

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

    @OneToOne
    @JoinColumn(name = "user_id", referencedColumnName = "id")
    private User user;

    @Column(nullable = false, unique = true, length = 500)
    private String token;

    @Column(nullable = false)
    private Instant expiryDate;

    @Column(name = "revoked")
    @Builder.Default
    private boolean revoked = false;

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

    @Column(name = "ip_address")
    private String ipAddress;

    @Column(name = "user_agent")
    private String userAgent;
}
```

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

==

```java
package com.lostandfound.model;

import java.time.LocalDateTime;

import org.hibernate.annotations.CreationTimestamp;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.EnumType;
import jakarta.persistence.Enumerated;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.Table;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Entity
@Table(name = "users")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class User {

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

    @Column(nullable = false)
    private String name;

    @Column(nullable = false, unique = true)
    private String email;

    @Column(nullable = false)
    private String password;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false)
    private Role role = Role.USER;

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

    public enum Role {
        USER, ADMIN
    }
}
```

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

==

```java
package com.lostandfound.repository;

import java.time.LocalDateTime;
import java.util.List;
import java.util.Optional;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import com.lostandfound.model.Claim;
import com.lostandfound.model.Item;
import com.lostandfound.model.User;

@Repository
public interface ClaimRepository extends JpaRepository<Claim, Long> {

    List<Claim> findByClaimedByOrderByClaimedAtDesc(User user);

    Optional<Claim> findByItemAndClaimedBy(Item item, User user);

    boolean existsByItemAndClaimedBy(Item item, User user);

    @Query("SELECT c FROM Claim c WHERE c.claimedAt < :cutoffDate")
    List<Claim> findOldClaims(@Param("cutoffDate") LocalDateTime cutoffDate);

    List<Claim> findByItem(Item item);

    List<Claim> findAllByOrderByClaimedAtDesc();
}
```

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

==

```java
package com.lostandfound.repository;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.lostandfound.model.Feedback;

@Repository
public interface FeedbackRepository extends JpaRepository<Feedback, Long> {
    List<Feedback> findAllByOrderBySubmittedAtDesc();
}
```

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

==

```java
package com.lostandfound.repository;

import com.lostandfound.model.Item;
import com.lostandfound.model.Item.Status;
import com.lostandfound.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;
import org.springframework.stereotype.Repository;

import java.util.List;

@Repository
public interface ItemRepository extends JpaRepository<Item, Long> {

    List<Item> findByCreatedByOrderByCreatedAtDesc(User user);

    @Query("SELECT i FROM Item i WHERE " +
           "(:search IS NULL OR :search = '' OR " +
           "LOWER(i.name) LIKE LOWER(CONCAT('%', :search, '%')) OR " +
           "LOWER(i.description) LIKE LOWER(CONCAT('%', :search, '%')) OR " +
           "LOWER(i.location) LIKE LOWER(CONCAT('%', :search, '%'))) AND " +
           "(:status IS NULL OR i.status = :status) " +
           "ORDER BY i.createdAt DESC")
    List<Item> searchItems(@Param("search") String search,
                           @Param("status") Status status);

    List<Item> findAllByOrderByCreatedAtDesc();
}
```

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

==

```java
package com.lostandfound.repository;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import com.lostandfound.model.Message;
import com.lostandfound.model.User;

@Repository
public interface MessageRepository extends JpaRepository<Message, Long> {
    List<Message> findByReceiverOrderBySentAtDesc(User receiver);
    List<Message> findBySenderOrderBySentAtDesc(User sender);
    List<Message> findAllByOrderBySentAtDesc();
}
```

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

==

```java
package com.lostandfound.repository;

import com.lostandfound.model.RefreshToken;
import com.lostandfound.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Modifying;
import org.springframework.data.jpa.repository.Query;
import org.springframework.stereotype.Repository;

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

@Repository
public interface RefreshTokenRepository extends JpaRepository<RefreshToken, Long> {

    Optional<RefreshToken> findByToken(String token);

    Optional<RefreshToken> findByUser(User user);

    @Modifying
    @Query("DELETE FROM RefreshToken rt WHERE rt.user = ?1")
    int deleteByUser(User user);

    @Modifying
    @Query("DELETE FROM RefreshToken rt WHERE rt.expiryDate < ?1")
    int deleteAllExpiredTokens(Instant now);

    @Query("SELECT rt FROM RefreshToken rt WHERE rt.user = ?1 AND rt.revoked = false")
    List<RefreshToken> findActiveTokensByUser(User user);

    boolean existsByToken(String token);
}
```

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

==

```java
package com.lostandfound.repository;

import com.lostandfound.model.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

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

@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByEmail(String email);
    Boolean existsByEmail(String email);
    List<User> findByRoleNot(User.Role role);
}
```

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

==

```java
package com.lostandfound.scheduler;

import com.lostandfound.model.Claim;
import com.lostandfound.model.Item;
import com.lostandfound.repository.ClaimRepository;
import com.lostandfound.repository.ItemRepository;
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

import java.time.LocalDateTime;
import java.util.List;

@Component
@RequiredArgsConstructor
public class ClaimCleanupScheduler {

    private static final Logger logger = LoggerFactory.getLogger(ClaimCleanupScheduler.cla
| ss);

    private final ClaimRepository claimRepository;
    private final ItemRepository itemRepository;

    @Scheduled(cron = "0 0 2 * * ?") // Run every day at 2 AM
    @Transactional
    public void cleanupOldClaims() {
        logger.info("Starting cleanup of old claims...");

        try {
            LocalDateTime cutoffDate = LocalDateTime.now().minusDays(7);
            List<Claim> oldClaims = claimRepository.findOldClaims(cutoffDate);

            int claimsDeleted = oldClaims.size();

            for (Claim claim : oldClaims) {
                Item item = claim.getItem();
                claimRepository.delete(claim);

                // Update item status if it was claimed
                if (item.getStatus() == Item.Status.CLAIMED) {
                    item.setStatus(Item.Status.FOUND);
                    itemRepository.save(item);
                }
            }

            logger.info("Deleted {} old claims and updated item statuses", claimsDeleted);
        } catch (Exception e) {
            logger.error("Error during claim cleanup", e);
        }
    }
}

----------------------------------------------------------------------------------------------
```

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

```
==============================================================================================
package com.lostandfound.scheduler;

import com.lostandfound.service.RefreshTokenService;
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;

@Component
@RequiredArgsConstructor
public class TokenCleanupScheduler {

    private static final Logger logger = LoggerFactory.getLogger(TokenCleanupScheduler.cla
   | ss);

    private final RefreshTokenService refreshTokenService;

    // Run every day at 3 AM
    @Scheduled(cron = "0 0 3 * * ?")
    @Transactional
    public void cleanupExpiredTokens() {
        logger.info("Starting cleanup of expired refresh tokens...");

        try {
            refreshTokenService.deleteExpiredTokens();
            logger.info("Successfully cleaned up expired refresh tokens");
        } catch (Exception e) {
            logger.error("Error during token cleanup", e);
        }
    }
}

----------------------------------------------------------------------------------------------
```

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

```
==============================================================================================
package com.lostandfound.security;

import com.lostandfound.exception.ResourceNotFoundException;
import com.lostandfound.model.User;
import com.lostandfound.repository.UserRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
@RequiredArgsConstructor
public class CustomUserDetailsService implements UserDetailsService {

    private final UserRepository userRepository;

    @Override
    @Transactional
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
        User user = userRepository.findByEmail(email)
                .orElseThrow(() -> new UsernameNotFoundException("User not found with emai
   | l: " + email));

```

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

    @Transactional
    public UserDetails loadUserById(Long id) {
        User user = userRepository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", id));

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

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

```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);
        }
        return null;
    }
}
```

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

===

```java
package com.lostandfound.security;

import io.jsonwebtoken.*;
import io.jsonwebtoken.security.Keys;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.core.Authentication;
import org.springframework.stereotype.Component;

import javax.crypto.SecretKey;
import java.util.Date;

@Component
public class JwtTokenProvider {

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

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

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

    private SecretKey getSigningKey() {
        return Keys.hmacShaKeyFor(jwtSecret.getBytes());
    }

    public String generateToken(Authentication authentication) {
        UserPrincipal userPrincipal = (UserPrincipal) authentication.getPrincipal();

        Date now = new Date();
        Date expiryDate = new Date(now.getTime() + jwtExpirationMs);

        return Jwts.builder()
                .subject(Long.toString(userPrincipal.getId()))
                .claim("email", userPrincipal.getEmail())
                .claim("role", userPrincipal.getAuthorities().iterator().next().getAuthority())
                .issuedAt(now)
                .expiration(expiryDate)
                .signWith(getSigningKey())
                .compact();
    }

    public Long getUserIdFromToken(String token) {
        Claims claims = Jwts.parser()
                .verifyWith(getSigningKey())
                .build()
                .parseSignedClaims(token)
                .getPayload();

        return Long.parseLong(claims.getSubject());
    }

    public boolean validateToken(String token) {
        try {
            Jwts.parser()
                .verifyWith(getSigningKey())
                .build()
                .parseSignedClaims(token);
            return true;
        } catch (JwtException | IllegalArgumentException e) {
            logger.error("Invalid JWT token: {}", e.getMessage());
            return false;
        }
    }
}
```

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

```java
package com.lostandfound.security;

import com.fasterxml.jackson.databind.ObjectMapper;
import com.lostandfound.config.RateLimitConfig;
import com.lostandfound.dto.response.ApiResponse;
import io.github.bucket4j.Bucket;
import io.github.bucket4j.ConsumptionProbe;
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.beans.factory.annotation.Value;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.stereotype.Component;
import org.springframework.web.filter.OncePerRequestFilter;

import java.io.IOException;

@Component
@RequiredArgsConstructor
public class RateLimitFilter extends OncePerRequestFilter {

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

    private final RateLimitConfig rateLimitConfig;
    private final ObjectMapper objectMapper;

    @Value("${rate.limit.enabled:true}")
    private boolean rateLimitEnabled;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {

        // Skip rate limiting if disabled
        if (!rateLimitEnabled) {
            filterChain.doFilter(request, response);
            return;
        }

        String path = request.getRequestURI();
        String clientIp = getClientIP(request);

        // Determine rate limit type based on path
        RateLimitConfig.RateLimitType limitType = determineRateLimitType(path);

        // Create unique key: IP + Path pattern
        String bucketKey = clientIp + ":" + limitType.name();

        // Get or create bucket for this key
        Bucket bucket = rateLimitConfig.resolveBucket(bucketKey, limitType);

        // Try to consume a token
        ConsumptionProbe probe = bucket.tryConsumeAndReturnRemaining(1);

        if (probe.isConsumed()) {
            // Request allowed - add rate limit headers
            response.addHeader("X-Rate-Limit-Remaining", String.valueOf(probe.getRemainingTokens()));
            response.addHeader("X-Rate-Limit-Retry-After-Seconds",
                    String.valueOf(limitType.getRefillDuration().getSeconds()));

            filterChain.doFilter(request, response);
        } else {
            // Rate limit exceeded
            long waitForRefill = probe.getNanosToWaitForRefill() / 1_000_000_000;

            logger.warn("Rate limit exceeded for IP: {} on path: {}", clientIp, path);

            response.setStatus(HttpStatus.TOO_MANY_REQUESTS.value());
            response.setContentType(MediaType.APPLICATION_JSON_VALUE);
            response.addHeader("X-Rate-Limit-Retry-After-Seconds", String.valueOf(waitForRefill));

            ApiResponse apiResponse = ApiResponse.builder()
                    .success(false)
                    .message("Rate limit exceeded. Please try again in " + waitForRefill +
|  " seconds.")
                    .build();

            response.getWriter().write(objectMapper.writeValueAsString(apiResponse));
        }
    }

    /**
     * Determine rate limit type based on request path
     */
    private RateLimitConfig.RateLimitType determineRateLimitType(String path) {
        if (path.startsWith("/api/auth/")) {
            return RateLimitConfig.RateLimitType.AUTH;
        } else if (path.startsWith("/admin/")) {
            return RateLimitConfig.RateLimitType.ADMIN;
        } else if (path.startsWith("/items") && path.contains("/image")) {
            return RateLimitConfig.RateLimitType.UPLOAD;
        } else if (path.startsWith("/uploads/") || path.startsWith("/static/")) {
            return RateLimitConfig.RateLimitType.PUBLIC;
        } else {
            return RateLimitConfig.RateLimitType.API;
        }
    }

    /**
     * Extract client IP address with proper proxy handling
     */
    private String getClientIP(HttpServletRequest request) {
        // Only trust X-Forwarded-For if behind a known proxy
        String xfHeader = request.getHeader("X-Forwarded-For");

        // Check if request is from trusted proxy
        String remoteAddr = request.getRemoteAddr();
        boolean isTrustedProxy = isTrustedProxy(remoteAddr);

        if (isTrustedProxy && xfHeader != null && !xfHeader.isEmpty()) {
            // X-Forwarded-For can contain multiple IPs, get the first one
            String clientIp = xfHeader.split(",")[0].trim();

            // Validate IP format to prevent injection
            if (isValidIP(clientIp)) {
                return clientIp;
            }
        }

        return remoteAddr;
    }

    /**
     * Check if IP is from trusted proxy
     */
    private boolean isTrustedProxy(String ip) {
        // Add your trusted proxy IPs here (e.g., load balancer, reverse proxy)
        // For local development, trust localhost
        return "127.0.0.1".equals(ip) ||
                "0:0:0:0:0:0:0:1".equals(ip) ||
                "::1".equals(ip);
        // In production, add your actual proxy IPs:
        // return Arrays.asList("10.0.0.1", "172.16.0.1").contains(ip);
    }

    /**
     * Validate IP address format
     */
    private boolean isValidIP(String ip) {
        if (ip == null || ip.isEmpty()) {
            return false;
        }

        // Simple validation for IPv4
        String ipv4Pattern = "^((25[0-5]|(2[0-4]|1\\d|[1-9]|)\\d)\\.?\\b){4}$";

        // Simple validation for IPv6
        String ipv6Pattern = "^(([0-9a-fA-F]{1,4}:){7}[0-9a-fA-F]{1,4}|" +
                "([0-9a-fA-F]{1,4}:){1,7}:|" +
                "([0-9a-fA-F]{1,4}:){1,6}:[0-9a-fA-F]{1,4}|" +
                "([0-9a-fA-F]{1,4}:){1,5}(:[0-9a-fA-F]{1,4}){1,2})$";

        return ip.matches(ipv4Pattern) || ip.matches(ipv6Pattern);
    }

    /**
     * Skip rate limiting for health check endpoints
     */
    @Override
```

```java
    protected boolean shouldNotFilter(HttpServletRequest request) {
        String path = request.getRequestURI();
        return path.startsWith("/actuator/health") ||
                path.startsWith("/actuator/info");
    }
}
```

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

==

```java
package com.lostandfound.security;

import com.lostandfound.model.User;
import lombok.AllArgsConstructor;
import lombok.Data;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;

import java.util.Collection;
import java.util.Collections;

@Data
@AllArgsConstructor
public class UserPrincipal implements UserDetails {

    private Long id;
    private String name;
    private String email;
    private String password;
    private Collection<? extends GrantedAuthority> authorities;

    public static UserPrincipal create(User user) {
        Collection<GrantedAuthority> authorities = Collections.singleton(
            new SimpleGrantedAuthority("ROLE_" + user.getRole().name())
        );

        return new UserPrincipal(
            user.getId(),
            user.getName(),
            user.getEmail(),
            user.getPassword(),
            authorities
        );
    }

    @Override
    public String getUsername() {
        return email;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

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

==

```java
package com.lostandfound.service;

import com.lostandfound.dto.response.ApiResponse;
import com.lostandfound.dto.response.ClaimResponse;
import com.lostandfound.exception.BadRequestException;
import com.lostandfound.exception.ResourceNotFoundException;
import com.lostandfound.exception.UnauthorizedException;
import com.lostandfound.model.Claim;
import com.lostandfound.model.Item;
import com.lostandfound.model.User;
import com.lostandfound.repository.ClaimRepository;
import com.lostandfound.repository.ItemRepository;
import com.lostandfound.repository.UserRepository;
import com.lostandfound.security.UserPrincipal;
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

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

@Service
@RequiredArgsConstructor
public class ClaimService {

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

    private final ClaimRepository claimRepository;
    private final ItemRepository itemRepository;
    private final UserRepository userRepository;

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

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

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

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

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

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

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

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

            claimRepository.save(claim);

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

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

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

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

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

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

    @Transactional
    public void deleteClaim(Long claimId) {
        Claim claim = claimRepository.findById(claimId)
                .orElseThrow(() -> new ResourceNotFoundException("Claim", "id", claimId));

        Item item = claim.getItem();

        claimRepository.delete(claim);

        // Update item status back to FOUND if it was CLAIMED and no other claims exist
        if (item.getStatus() == Item.Status.CLAIMED) {
            List<Claim> remainingClaims = claimRepository.findByItem(item);
            if (remainingClaims.isEmpty()) {
                item.setStatus(Item.Status.FOUND);
                itemRepository.save(item);
            }
        }

        logger.info("Deleted claim {} for item {}", claimId, item.getId());
    }

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

        Claim claim = claimRepository.findById(claimId)
                .orElseThrow(() -> new ResourceNotFoundException("Claim", "id", claimId));

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

        boolean isAdmin = user.getRole() == User.Role.ADMIN;
        boolean isOwner = claim.getClaimedBy().getId().equals(currentUser.getId());

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

        deleteClaim(claimId);
    }

    private ClaimResponse mapToClaimResponse(Claim claim) {
        return ClaimResponse.builder()
                .id(claim.getId())
                .itemId(claim.getItem().getId())
                .itemName(claim.getItem().getName())
                .description(claim.getItem().getDescription())
                .location(claim.getItem().getLocation())
                .claimedBy(claim.getClaimedBy().getId())
                .claimerName(claim.getClaimedBy().getName())
                .claimantName(claim.getClaimantName())
                .claimantEmail(claim.getClaimantEmail())
                .claimedAt(claim.getClaimedAt())
                .build();
    }
}
```

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

==

```java
package com.lostandfound.service;

import com.lostandfound.dto.request.FeedbackRequest;
import com.lostandfound.dto.response.ApiResponse;
import com.lostandfound.dto.response.FeedbackResponse;
import com.lostandfound.exception.ResourceNotFoundException;
import com.lostandfound.model.Feedback;
import com.lostandfound.model.User;
import com.lostandfound.repository.FeedbackRepository;
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 java.util.List;
import java.util.stream.Collectors;

@Service
@RequiredArgsConstructor
public class FeedbackService {

    private final FeedbackRepository feedbackRepository;
    private final UserRepository userRepository;

    @Transactional
    public ApiResponse submitFeedback(FeedbackRequest request, UserPrincipal currentUser)
 | {
        User user = userRepository.findById(currentUser.getId())
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
 | .getId()));

        Feedback feedback = new Feedback();
        feedback.setUser(user);
        feedback.setFeedbackText(request.getFeedback());

        feedbackRepository.save(feedback);

        return ApiResponse.builder()
                .success(true)
                .message("Thank you for your feedback!")
                .build();
    }

    @Transactional(readOnly = true)
    public List<FeedbackResponse> getAllFeedback() {
        List<Feedback> feedbacks = feedbackRepository.findAllByOrderBySubmittedAtDesc();
        return feedbacks.stream()
                .map(this::mapToFeedbackResponse)
                .collect(Collectors.toList());
    }

    @Transactional
    public void deleteFeedback(Long feedbackId) {
        Feedback feedback = feedbackRepository.findById(feedbackId)
                .orElseThrow(() -> new ResourceNotFoundException("Feedback", "id", feedbac
 | kId));

        feedbackRepository.delete(feedback);
    }

    private FeedbackResponse mapToFeedbackResponse(Feedback feedback) {
        return FeedbackResponse.builder()
                .id(feedback.getId())
                .userId(feedback.getUser().getId())
                .userName(feedback.getUser().getName())
                .feedbackText(feedback.getFeedbackText())
                .submittedAt(feedback.getSubmittedAt())
                .build();
    }
}
```

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

==

```java
package com.lostandfound.service;

import com.lostandfound.config.FileStorageProperties;
import com.lostandfound.exception.BadRequestException;
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Service;
import org.springframework.util.StringUtils;
import org.springframework.web.multipart.MultipartFile;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardCopyOption;
import java.util.UUID;

@Service
@RequiredArgsConstructor
public class FileStorageService {

    private static final Logger logger = LoggerFactory.getLogger(FileStorageService.class);
    private final FileStorageProperties fileStorageProperties;

    public String storeFile(MultipartFile file) {
        // Normalize file name
        String originalFileName = StringUtils.cleanPath(file.getOriginalFilename());

        try {
            // Check if the file's name contains invalid characters
            if (originalFileName.contains("..")) {
                throw new BadRequestException("Filename contains invalid path sequence " + originalFileName);
            }

            // Create unique filename
            String fileExtension = "";
            if (originalFileName.contains(".")) {
                fileExtension = originalFileName.substring(originalFileName.lastIndexOf("."));
            }
            String uniqueFileName = UUID.randomUUID().toString() + "_" + originalFileName;

            // 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_EXISTING);

            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/", "");

                // Prevent path traversal
                if (fileName.contains("..") || fileName.contains("/") || fileName.contains("\\")) {
                    logger.error("Invalid file path detected: {}", filePath);
                    return;
                }

                Path fileStorageLocation = Paths.get(fileStorageProperties.getUploadDir())
                        .toAbsolutePath().normalize();
                Path targetLocation = fileStorageLocation.resolve(fileName).normalize();

                // Ensure the resolved path is still within the upload directory
                if (!targetLocation.startsWith(fileStorageLocation)) {
                    logger.error("Path traversal attempt detected: {}", filePath);
                    return;
                }

                Files.deleteIfExists(targetLocation);
            }
        } catch (IOException ex) {
            // Log error but don't throw exception
            logger.error("Could not delete file: {}", filePath, ex);
        }
    }
}
```

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

```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.exception.UnauthorizedException;
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.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.multipart.MultipartFile;

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

@Service
@RequiredArgsConstructor
public class ItemService {

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

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

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

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

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

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

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

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

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

        return mapToItemResponse(item);
    }

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

        return mapToItemResponse(item);
    }

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

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

        List<Item> items = itemRepository.searchItems(
                sanitizedSearch,
                itemStatus
        );

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

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

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

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

        return sanitized;
    }

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

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

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

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

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

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

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

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

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

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

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

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

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

        return mapToItemResponse(item);
    }

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    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

===

```java
package com.lostandfound.service;

import com.lostandfound.dto.request.MessageRequest;
import com.lostandfound.dto.response.ApiResponse;
import com.lostandfound.dto.response.MessageResponse;
import com.lostandfound.exception.BadRequestException;
import com.lostandfound.exception.ResourceNotFoundException;
import com.lostandfound.exception.UnauthorizedException;
import com.lostandfound.model.Item;
import com.lostandfound.model.Message;
import com.lostandfound.model.User;
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.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

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

@Service
@RequiredArgsConstructor
public class MessageService {

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

    private final MessageRepository messageRepository;
    private final UserRepository userRepository;
    private final ItemRepository itemRepository;

    @Transactional
    public ApiResponse sendMessage(MessageRequest request, UserPrincipal currentUser) {
        if (currentUser == null) {
            throw new UnauthorizedException("You must be logged in to send a message");
        }

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

        User receiver = userRepository.findById(request.getReceiverId())
                .orElseThrow(() -> new ResourceNotFoundException("Receiver user", "id", request.getReceiverId()));

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

        messageRepository.delete(message);
        logger.info("User {} deleted message {}", user.getEmail(), messageId);
    }

    private MessageResponse mapToMessageResponse(Message message) {
        return MessageResponse.builder()
                .id(message.getId())
                .senderId(message.getSender().getId())
                .senderName(message.getSender().getName())
                .receiverId(message.getReceiver().getId())
                .itemId(message.getItem().getId())
                .itemName(message.getItem().getName())
                .message(message.getMessage())
                .sentAt(message.getSentAt())
                .build();
    }
}
```

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

==

```java
package com.lostandfound.service;

import com.lostandfound.exception.BadRequestException;
import com.lostandfound.exception.UnauthorizedException;
import com.lostandfound.model.RefreshToken;
import com.lostandfound.model.User;
import com.lostandfound.repository.RefreshTokenRepository;
import com.lostandfound.repository.UserRepository;
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

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

@Service
@RequiredArgsConstructor
public class RefreshTokenService {

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

    private final RefreshTokenRepository refreshTokenRepository;
    private final UserRepository userRepository;

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

    @Transactional
    public RefreshToken createRefreshToken(Long userId, String ipAddress, String userAgent) {
        User user = userRepository.findById(userId)
                .orElseThrow(() -> new BadRequestException("User not found"));

        // Delete old refresh tokens for this user
        refreshTokenRepository.deleteByUser(user);

        RefreshToken refreshToken = RefreshToken.builder()
                .user(user)
                .token(UUID.randomUUID().toString())
                .expiryDate(Instant.now().plusMillis(refreshTokenDurationMs))
                .revoked(false)
                .ipAddress(ipAddress)
                .userAgent(userAgent)
                .build();

        logger.info("Created new refresh token for user: {}", user.getEmail());
        return refreshTokenRepository.save(refreshToken);
    }

    @Transactional(readOnly = true)
    public Optional<RefreshToken> findByToken(String token) {
        if (token == null || token.trim().isEmpty()) {
            return Optional.empty();
        }
        return refreshTokenRepository.findByToken(token.trim());
    }

    @Transactional
    public RefreshToken verifyExpiration(RefreshToken token) {
        if (token.getExpiryDate().compareTo(Instant.now()) < 0) {
            refreshTokenRepository.delete(token);
            logger.warn("Refresh token expired for user: {}", token.getUser().getEmail());
            throw new BadRequestException("Refresh token expired. Please login again.");
        }

        if (token.isRevoked()) {
            logger.warn("Attempted to use revoked refresh token for user: {}", token.getUser().getEmail());
            throw new BadRequestException("Refresh token has been revoked. Please login again.");
        }

        return token;
    }

    @Transactional
    public void revokeToken(String token) {
        if (token == null || token.trim().isEmpty()) {
            throw new BadRequestException("Invalid refresh token");
        }

        RefreshToken refreshToken = refreshTokenRepository.findByToken(token.trim())
                .orElseThrow(() -> new BadRequestException("Invalid refresh token"));

        if (refreshToken.isRevoked()) {
            throw new BadRequestException("Token is already revoked");
        }

        refreshToken.setRevoked(true);
        refreshTokenRepository.save(refreshToken);
        logger.info("Revoked refresh token for user: {}", refreshToken.getUser().getEmail());
    }

    @Transactional
    public void revokeTokenForUser(String token, Long userId) {
        if (token == null || token.trim().isEmpty()) {
            throw new BadRequestException("Invalid refresh token");
        }

        RefreshToken refreshToken = refreshTokenRepository.findByToken(token.trim())
                .orElseThrow(() -> new BadRequestException("Invalid refresh token"));

        // Verify the token belongs to the user
        if (!refreshToken.getUser().getId().equals(userId)) {
            logger.warn("User {} attempted to revoke token belonging to user {}",
                    userId, refreshToken.getUser().getId());
            throw new UnauthorizedException("This token does not belong to you");
        }

        if (refreshToken.isRevoked()) {
            throw new BadRequestException("Token is already revoked");
        }

        refreshToken.setRevoked(true);
        refreshTokenRepository.save(refreshToken);
        logger.info("Revoked refresh token for user: {}", refreshToken.getUser().getEmail());
    }

    @Transactional
    public void revokeAllUserTokens(Long userId) {
        User user = userRepository.findById(userId)
```

```java
                .orElseThrow(() -> new BadRequestException("User not found"));

        refreshTokenRepository.deleteByUser(user);
        logger.info("Revoked all refresh tokens for user: {}", user.getEmail());
    }

    @Transactional
    public void deleteExpiredTokens() {
        int deletedCount = refreshTokenRepository.deleteAllExpiredTokens(Instant.now());
        logger.info("Deleted {} expired refresh tokens", deletedCount);
    }
}
```

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

==

```java
package com.lostandfound.service;

import com.lostandfound.dto.request.LoginRequest;
import com.lostandfound.dto.request.RegisterRequest;
import com.lostandfound.dto.response.AuthResponse;
import com.lostandfound.dto.response.TokenRefreshResponse;
import com.lostandfound.exception.BadRequestException;
import com.lostandfound.exception.ResourceNotFoundException;
import com.lostandfound.model.RefreshToken;
import com.lostandfound.model.User;
import com.lostandfound.repository.UserRepository;
import com.lostandfound.security.JwtTokenProvider;
import com.lostandfound.security.UserPrincipal;
import lombok.RequiredArgsConstructor;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.LockedException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
@RequiredArgsConstructor
public class UserService {

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

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;
    private final AuthenticationManager authenticationManager;
    private final JwtTokenProvider tokenProvider;
    private final RefreshTokenService refreshTokenService;

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

    @Transactional
    public AuthResponse registerUser(RegisterRequest request, String ipAddress, String use
  | rAgent) {
        // Normalize email to lowercase
        String email = request.getEmail().toLowerCase().trim();

        // Check if email already exists
        if (userRepository.existsByEmail(email)) {
            logger.warn("Registration attempt with existing email");
            // Use generic message to prevent user enumeration
            // Add a small delay to prevent timing attacks
            try {
                Thread.sleep(100 + (long)(Math.random() * 200)); // 100-300ms random delay
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
            throw new BadRequestException("Registration failed. Please check your informat
  | ion and try again.");
        }

        // Validate password confirmation
        if (!request.getPassword().equals(request.getConfirmPassword())) {
            throw new BadRequestException("Passwords do not match");
        }

        // Create new user
        User user = new User();
        user.setName(request.getName().trim());
        user.setEmail(email);
        user.setPassword(passwordEncoder.encode(request.getPassword()));
        user.setRole(User.Role.USER);

        try {
            user = userRepository.save(user);
            logger.info("New user registered successfully with ID: {}", user.getId());
        } catch (Exception e) {
            logger.error("Error during user registration", e);
            throw new BadRequestException("Registration failed. Please try again.");
        }

        // Auto-login after registration
        Authentication authentication = authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(email, request.getPassword())
        );

        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 with ID: {}", user.getId());

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

    @Transactional
    public void logoutUser(Long userId) {
        User user = userRepository.findById(userId)
                .orElseThrow(() -> new BadRequestException("User not found"));

        // Delete all refresh tokens for this user
        refreshTokenService.revokeAllUserTokens(userId);

        logger.info("User logged out successfully with ID: {}", user.getId());
    }

    @Transactional
    public TokenRefreshResponse refreshToken(String refreshTokenStr) {
        return refreshTokenService.findByToken(refreshTokenStr)
                .map(refreshTokenService::verifyExpiration)
                .map(RefreshToken::getUser)
                .map(user -> {
                    UserPrincipal userPrincipal = UserPrincipal.create(user);
                    Authentication auth = new UsernamePasswordAuthenticationToken(
                            userPrincipal, null, userPrincipal.getAuthorities()
                    );

                    String newAccessToken = tokenProvider.generateToken(auth);

                    return TokenRefreshResponse.builder()
                            .success(true)
                            .message("Token refreshed successfully")
                            .accessToken(newAccessToken)
                            .refreshToken(refreshTokenStr)
                            .tokenType("Bearer")
                            .build();
                })
                .orElseThrow(() -> new BadRequestException("Invalid refresh token"));
    }

    @Transactional(readOnly = true)
    public User getUserById(Long userId) {
        return userRepository.findById(userId)
                .orElseThrow(() -> new BadRequestException("User not found"));
    }

    @Transactional(readOnly = true)
    public User getUserByEmail(String email) {
        return userRepository.findByEmail(email.toLowerCase().trim())
                .orElseThrow(() -> new BadRequestException("User not found"));
    }

    @Transactional
    public void deleteUser(Long userId, UserPrincipal currentUser) {
        // Verify current user is admin
        User adminUser = userRepository.findById(currentUser.getId())
                .orElseThrow(() -> new BadRequestException("Admin user not found"));

        if (adminUser.getRole() != User.Role.ADMIN) {
            throw new BadRequestException("Only admins can delete users");
        }

        // Find the user to delete
        User userToDelete = userRepository.findById(userId)
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", userId));

        // Prevent deleting admin users
        if (userToDelete.getRole() == User.Role.ADMIN) {
            throw new BadRequestException("Cannot delete admin user");
        }
```

```
236:
237:         logger.info("Admin ID {} deleting user ID {}", adminUser.getId(), userToDelete.get
    | Id());
238:
239:         // Delete all refresh tokens for this user
240:         refreshTokenService.revokeAllUserTokens(userId);
241:
242:         // Delete the user (cascading will handle related data if configured)
243:         userRepository.delete(userToDelete);
244:
245:         logger.info("Successfully deleted user ID: {}", userToDelete.getId());
246:     }
247: }
```

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

===

```
  1: package com.lostandfound.util;
  2:
  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);
 44:     }
 45:
 46:     /**
 47:      * Generic method to add cookie
 48:      */
 49: // In the addCookie method, update line 59-67:
 50:     private void addCookie(HttpServletResponse response, String name, String value,
 51:                            int maxAge, boolean httpOnly) {
 52:         Cookie cookie = new Cookie(name, value);
 53:         cookie.setPath("/");
 54:         cookie.setHttpOnly(httpOnly);
 55:         cookie.setMaxAge(maxAge);
 56:         cookie.setSecure(cookieSecure);
 57:
 58:         // SameSite attribute - build proper header
 59:         StringBuilder cookieHeader = new StringBuilder();
 60:         cookieHeader.append(name).append("=").append(value);
 61:         cookieHeader.append("; Path=/");
 62:         cookieHeader.append("; Max-Age=").append(maxAge);
 63:
 64:         if (httpOnly) {
 65:             cookieHeader.append("; HttpOnly");
 66:         }
 67:
```

```java
        if (cookieSecure) {
            cookieHeader.append("; Secure");
        }

        // SameSite - only add if specified
        if (cookieSameSite != null && !cookieSameSite.isEmpty()) {
            cookieHeader.append("; SameSite=").append(cookieSameSite);
        }

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

    /**
     * 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);
    }
}
```
