■ 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\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\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\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\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\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\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\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\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\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\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\UserService.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\config\FileStorageProperties.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

```
package com.lostandfound.config;

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\SecurityConfig.java

```
package com.lostandfound.config;

import com.lostandfound.security.JwtAuthenticationFilter;
import lombok.RequiredArgsConstructor;
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;
```

```java
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
@RequiredArgsConstructor
public class SecurityConfig {

    private final JwtAuthenticationFilter jwtAuthenticationFilter;
    private final UserDetailsService userDetailsService;

    @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 {
        http
                .csrf(AbstractHttpConfigurer::disable)
                .cors(cors -> cors.configurationSource(corsConfigurationSource()))
                .sessionManagement(session ->
                        session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
                .authorizeHttpRequests(auth -> auth
                        .requestMatchers(
                                "/api/auth/**",
                                "/uploads/**",
                                "/static/**",
                                "/error"
                        ).permitAll()

                        .requestMatchers("/admin/**").hasRole("ADMIN")
                        .anyRequest().authenticated()
                )
                .authenticationProvider(authenticationProvider())
                .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFilter.class)
                // Security headers
                .headers(headers -> headers
                        .contentSecurityPolicy(csp ->
                                csp.policyDirectives("default-src 'self'"))
                        .frameOptions(frame -> frame.deny())
                        .xssProtection(xss -> xss.headerValue(XXssProtectionHeaderWriter.HeaderValue.ENABLED_MODE_BLOCK))
                );

        return http.build();
    }

    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        CorsConfiguration configuration = new CorsConfiguration();
        configuration.setAllowedOriginPatterns(Arrays.asList("http://localhost:*", "https://yourdomain.com"));
        configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE", "OPTIONS"));
        configuration.setAllowedHeaders(List.of("*"));
        configuration.setAllowCredentials(true);
        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

```
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

```
package com.lostandfound.controller;

import com.lostandfound.dto.response.ApiResponse;
import com.lostandfound.dto.response.ClaimResponse;
import com.lostandfound.dto.response.FeedbackResponse;
import com.lostandfound.dto.response.ItemResponse;
import com.lostandfound.exception.ResourceNotFoundException;
import com.lostandfound.model.User;
import com.lostandfound.repository.UserRepository;
import com.lostandfound.security.UserPrincipal;
import com.lostandfound.service.ClaimService;
import com.lostandfound.service.FeedbackService;
import com.lostandfound.service.ItemService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.security.core.annotation.AuthenticationPrincipal;
import org.springframework.web.bind.annotation.*;

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

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

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

    @GetMapping("/dashboard")
    public ResponseEntity<Map<String, Object>> getAdminDashboard() {
```

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

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

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

        return ResponseEntity.ok(response);
    }

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

        itemService.deleteItem(itemId, currentUser);

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

        return ResponseEntity.ok(response);
    }

    @DeleteMapping("/claims/{claimId}")
    public ResponseEntity<ApiResponse> deleteClaim(@PathVariable Long claimId) {
        claimService.deleteClaim(claimId);

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

        return ResponseEntity.ok(response);
    }

    @DeleteMapping("/users/{userId}")
    public ResponseEntity<ApiResponse> deleteUser(@PathVariable Long userId) {
        User user = userRepository.findById(userId)
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", userId));

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

        userRepository.delete(user);

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

        return ResponseEntity.ok(response);
    }

    @DeleteMapping("/feedback/{feedbackId}")
    public ResponseEntity<ApiResponse> deleteFeedback(@PathVariable Long feedbackId) {
        feedbackService.deleteFeedback(feedbackId);

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

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

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

==

```java
package com.lostandfound.controller;

import com.lostandfound.dto.response.ApiResponse;
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.*;

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

    private final ClaimService claimService;

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

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

■ 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.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) {

        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.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.*;

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

    private final FeedbackService feedbackService;

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

        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.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) {

        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, List<ItemResponse>>> getItems(
            @RequestParam(required = false, defaultValue = "") String search,
            @RequestParam(required = false, defaultValue = "") String status) {

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

        Map<String, List<ItemResponse>> response = new HashMap<>();
        response.put("items", items);

        return ResponseEntity.ok(response);
    }

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

        itemService.deleteItem(itemId, currentUser);

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

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

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

==

```
package com.lostandfound.controller;

import com.lostandfound.dto.request.MessageRequest;
import com.lostandfound.dto.response.ApiResponse;
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.*;

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

    private final MessageService messageService;

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

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

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

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

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

==

```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

==

```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

==

```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

==

```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
```

```java
@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\response\ApiResponse.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 ApiResponse {
    private boolean success;
    private String message;
    private Object data;
}
```

■ File: lost-and-found-backend\src\main\java\com\lostandfound\dto\response\AuthResponse.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 AuthResponse {
    private boolean success;
    private String message;
    private String token;
    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

==
```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

```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

```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\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) {

        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

==

```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
```

```java
@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

===

```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;
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 = "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;

    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\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> findAllByOrderBySentAtDesc();
}
```

■ 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\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));

        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 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;

    @Override
    protected void doFilterInternal(HttpServletRequest request,
                                    HttpServletResponse response,
                                    FilterChain filterChain) throws ServletException, IOException {
        try {
            String jwt = getJwtFromRequest(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);
            }
        } catch (Exception ex) {
            logger.error("Could not set user authentication in security context", ex);
        }

        filterChain.doFilter(request, response);
    }

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

■ File: 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
```

```java
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\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.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.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

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

@Service
@RequiredArgsConstructor
public class ClaimService {

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

    @Transactional
    public ApiResponse claimItem(Long itemId, UserPrincipal currentUser) {
        Item item = itemRepository.findById(itemId)
                .orElseThrow(() -> new ResourceNotFoundException("Item", "id", itemId));

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

        if (item.getStatus() == Item.Status.CLAIMED) {
            throw new BadRequestException("Item already claimed");
        }

        if (claimRepository.existsByItemAndClaimedBy(item, user)) {
            throw new BadRequestException("You have already claimed this item");
        }

        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);

        return ApiResponse.builder()
                .success(true)
                .message("Item claimed successfully")
                .build();
    }

    @Transactional(readOnly = true)
    public List<ClaimResponse> getUserClaims(UserPrincipal currentUser) {
        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
        if (item.getStatus() == Item.Status.CLAIMED) {
            item.setStatus(Item.Status.FOUND);
            itemRepository.save(item);
        }
    }

    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
```

```java
@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.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 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/", "");
                Path fileStorageLocation = Paths.get(fileStorageProperties.getUploadDir())
                        .toAbsolutePath().normalize();
                Path targetLocation = fileStorageLocation.resolve(fileName);
                Files.deleteIfExists(targetLocation);
            }
        } catch (IOException ex) {
            // Log error but don't throw exception
            System.err.println("Could not delete file: " + filePath);
        }
    }
}
```

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

```java
package com.lostandfound.service;

import com.lostandfound.dto.request.ItemRequest;
import com.lostandfound.dto.response.ItemResponse;
import com.lostandfound.exception.BadRequestException;
import com.lostandfound.exception.ResourceNotFoundException;
import com.lostandfound.model.Item;
import com.lostandfound.model.User;
import com.lostandfound.repository.ClaimRepository;
import com.lostandfound.repository.ItemRepository;
import com.lostandfound.repository.MessageRepository;
import com.lostandfound.repository.UserRepository;
import com.lostandfound.security.UserPrincipal;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import org.springframework.web.multipart.MultipartFile;

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

@Service
@RequiredArgsConstructor
public class ItemService {

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

    @Transactional
    public ItemResponse createItem(ItemRequest request, MultipartFile image, UserPrincipal currentUser) {
        User user = userRepository.findById(currentUser.getId())
```

```java
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser.getId()));

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

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

        item = itemRepository.save(item);

        return mapToItemResponse(item);
    }

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

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

    @Transactional(readOnly = true)
    public List<ItemResponse> getUserItems(UserPrincipal currentUser) {
        User user = userRepository.findById(currentUser.getId())
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser.getId()));

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

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

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

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

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

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

        itemRepository.delete(item);
    }

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

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

===

```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.ResourceNotFoundException;
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.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

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

@Service
@RequiredArgsConstructor
public class MessageService {

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

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

        User receiver = userRepository.findById(request.getReceiverId())
                .orElseThrow(() -> new ResourceNotFoundException("User", "id", request.get
| ReceiverId()));

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

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

        messageRepository.save(message);

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

    @Transactional(readOnly = true)
    public List<MessageResponse> getUserMessages(UserPrincipal currentUser) {
        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());
    }

    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\UserService.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.exception.BadRequestException;
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.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;

    @Transactional
    public AuthResponse registerUser(RegisterRequest request) {
        // 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: {}", email);
            throw new BadRequestException("Email address is already registered");
        }

        // 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: {}", email);
        } catch (Exception e) {
            logger.error("Error during user registration: {}", email, e);
            throw new BadRequestException("Registration failed. Please try again.");
        }

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

```

```java
        SecurityContextHolder.getContext().setAuthentication(authentication);
        String jwt = tokenProvider.generateToken(authentication);

        // 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")
            .token(jwt)
            .user(userDTO)
            .build();
    }

    @Transactional(readOnly = true)
    public AuthResponse loginUser(LoginRequest request) {
        String email = request.getEmail().toLowerCase().trim();

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

            SecurityContextHolder.getContext().setAuthentication(authentication);

            // Generate JWT token
            String jwt = tokenProvider.generateToken(authentication);

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

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

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

            return AuthResponse.builder()
                .success(true)
                .message("Login successful")
                .token(jwt)
                .user(userDTO)
                .build();

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

    @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"));
    }
}
```
