

# ■ 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

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

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

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

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

```
=====
1: package com.lostandfound.config;
2:
3: import org.modelmapper.ModelMapper;
4: import org.modelmapper.convention.MatchingStrategies;
5: import org.springframework.context.annotation.Bean;
6: import org.springframework.context.annotation.Configuration;
7:
8: @Configuration
9: public class ModelMapperConfig {
10:
11:     @Bean
12:     public ModelMapper modelMapper() {
13:         ModelMapper modelMapper = new ModelMapper();
14:         modelMapper.getConfiguration()
15:             .setMatchingStrategy(MatchingStrategies.STRICT)
16:             .setSkipNullEnabled(true);
17:         return modelMapper;
18:     }
19: }
```

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

```
=====
1: package com.lostandfound.config;
2:
3: import com.lostandfound.security.JwtAuthenticationFilter;
4: import lombok.RequiredArgsConstructor;
5: import org.springframework.context.annotation.Bean;
6: import org.springframework.context.annotation.Configuration;
7: import org.springframework.security.authentication.AuthenticationManager;
8: import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
9: import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
10: import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
11: import org.springframework.security.config.annotation.web.builders.HttpSecurity;
12: import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
13: import org.springframework.security.config.annotation.web.configurers.AbstractHttpConfigurer;
14: import org.springframework.security.config.http.SessionCreationPolicy;
15: import org.springframework.security.core.userdetails.UserDetailsService;
16: import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
17: import org.springframework.security.crypto.password.PasswordEncoder;
18: import org.springframework.security.web.SecurityFilterChain;
19: import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
20:
21: @Configuration
22: @EnableWebSecurity
23: @RequiredArgsConstructor
24: public class SecurityConfig {
25:     private final AuthenticationConfiguration authenticationConfiguration;
26:     private final UserDetailsService userDetailsService;
27:     private final PasswordEncoder passwordEncoder;
28:
29:     @Bean
30:     public SecurityFilterChain securityFilterChain() throws Exception {
31:         HttpSecurity httpSecurity = HttpSecurity.with(authenticationConfiguration)
32:             .addFilter(jwtAuthenticationFilter())
33:             .authorizeHttpRequests(authorize -> authorizeRequests()
34:                 .anyRequest().authenticated())
35:             .sessionManagement(sessionManagement -> sessionManagementPolicy(SessionCreationPolicy.STATELESS))
36:             .passwordEncoder(passwordEncoder)
37:             .authenticationProvider(daoAuthenticationProvider())
38:             .build();
39:         return httpSecurity.build();
40:     }
41:
42:     private DaoAuthenticationProvider daoAuthenticationProvider() {
43:         DaoAuthenticationProvider daoAuthenticationProvider = new DaoAuthenticationProvider();
44:         daoAuthenticationProvider.setUserDetailsService(userDetailsService);
45:         daoAuthenticationProvider.setPasswordEncoder(passwordEncoder);
46:         return daoAuthenticationProvider;
47:     }
48:
49:     private JwtAuthenticationFilter jwtAuthenticationFilter() {
50:         JwtAuthenticationFilter jwtAuthenticationFilter = new JwtAuthenticationFilter();
51:         jwtAuthenticationFilter.setAuthenticationManager(authenticationConfiguration.getAuthenticationManager());
52:         return jwtAuthenticationFilter;
53:     }
54: }
```

```

20: import org.springframework.security.web.header.writers.XXssProtectionHeaderWriter;
21: import org.springframework.web.cors.CorsConfiguration;
22: import org.springframework.web.cors.CorsConfigurationSource;
23: import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
24:
25: import java.util.Arrays;
26: import java.util.List;
27:
28: @Configuration
29: @EnableWebSecurity
30: @EnableMethodSecurity
31: @RequiredArgsConstructor
32: public class SecurityConfig {
33:
34:     private final JwtAuthenticationFilter jwtAuthenticationFilter;
35:     private final UserDetailsService userDetailsService;
36:
37:     @Bean
38:     public PasswordEncoder passwordEncoder() {
39:         // Use strength 12 for production (default is 10)
40:         return new BCryptPasswordEncoder(12);
41:     }
42:
43:     @Bean
44:     public DaoAuthenticationProvider authenticationProvider() {
45:         DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
46:         authProvider.setUserDetailsService(userDetailsService);
47:         authProvider.setPasswordEncoder(passwordEncoder());
48:         authProvider.setHideUserNotFoundExceptions(true); // Security best practice
49:         return authProvider;
50:     }
51:
52:     @Bean
53:     public AuthenticationManager authenticationManager(AuthenticationConfiguration authCon
| fig)
54:         throws Exception {
55:         return authConfig.getAuthenticationManager();
56:     }
57:
58:     @Bean
59:     public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
60:         http
61:             .csrf(AbstractHttpConfigurer::disable)
62:             .cors(cors -> cors.configurationSource(corsConfigurationSource()))
63:             .sessionManagement(session ->
64:                 session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
65:             .authorizeHttpRequests(auth -> auth
66:                 .requestMatchers(
67:                     "/api/auth/**",
68:                     "/uploads/**",
69:                     "/static/**",
70:                     "/error"
71:                 ).permitAll()
72:                 .requestMatchers("/admin/**").hasRole("ADMIN")
73:                 .anyRequest().authenticated()
74:             )
75:             .authenticationProvider(authenticationProvider())
76:             .addFilterBefore(jwtAuthenticationFilter, UsernamePasswordAuthenticationFi
| lter.class)
77:
78:         // Security headers
79:         .headers(headers -> headers
80:             .contentSecurityPolicy(csp ->
81:                 csp.policyDirectives("default-src 'self'"))
82:             .frameOptions(frame -> frame.deny())
83:             .xssProtection(xss -> xss.headerValue(XXssProtectionHeaderWriter.H
| eaderValue.ENABLED_MODE_BLOCK))
84:         );
85:
86:         return http.build();
87:     }
88:
89:     @Bean
90:     public CorsConfigurationSource corsConfigurationSource() {
91:         CorsConfiguration configuration = new CorsConfiguration();
92:         configuration.setAllowedOriginPatterns(Arrays.asList("http://localhost:*", "https:
| //yourdomain.com"));
93:         configuration.setAllowedMethods(Arrays.asList("GET", "POST", "PUT", "DELETE", "OPT
| IONS"));
94:         configuration.setAllowedHeaders(List.of("*"));
95:         configuration.setAllowCredentials(true);
96:         configuration.setMaxAge(3600L);
97:
98:         UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
99:         source.registerCorsConfiguration("/*", configuration);

```

```

100:         return source;
101:     }
102: }

```

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

```

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

```

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

```

=====
1: package com.lostandfound.controller;
2:
3: import com.lostandfound.dto.response.ApiResponse;
4: import com.lostandfound.dto.response.ClaimResponse;
5: import com.lostandfound.dto.response.FeedbackResponse;
6: import com.lostandfound.dto.response.ItemResponse;
7: import com.lostandfound.exception.ResourceNotFoundException;
8: import com.lostandfound.model.User;
9: import com.lostandfound.repository.UserRepository;
10: import com.lostandfound.security.UserPrincipal;
11: import com.lostandfound.service.ClaimService;
12: import com.lostandfound.service.FeedbackService;
13: import com.lostandfound.service.ItemService;
14: import lombok.RequiredArgsConstructor;
15: import org.springframework.http.ResponseEntity;
16: import org.springframework.security.access.prepost.PreAuthorize;
17: import org.springframework.security.core.annotation.AuthenticationPrincipal;
18: import org.springframework.web.bind.annotation.*;
19:
20: import java.util.HashMap;
21: import java.util.List;
22: import java.util.Map;
23: import java.util.stream.Collectors;
24:
25: @RestController
26: @RequestMapping("/admin")
27: @PreAuthorize("hasRole('ADMIN')")
28: @RequiredArgsConstructor
29: public class AdminController {
30:
31:     private final ItemService itemService;
32:     private final ClaimService claimService;
33:     private final FeedbackService feedbackService;
34:     private final UserRepository userRepository;
35:
36:     @GetMapping("/dashboard")
37:     public ResponseEntity<Map<String, Object>> getAdminDashboard() {

```

```

38:         List<ItemResponse> items = itemService.searchItems("", "");
39:         List<ClaimResponse> claims = claimService.getUserClaims(null); // Get all claims
40:         List<FeedbackResponse> feedback = feedbackService.getAllFeedback();
41:         List<User> users = userRepository.findByRoleNot(User.Role.ADMIN);
42:
43:         List<Map<String, Object>> userList = users.stream()
44:             .map(user -> {
45:                 Map<String, Object> userMap = new HashMap<>();
46:                 userMap.put("id", user.getId());
47:                 userMap.put("name", user.getName());
48:                 userMap.put("email", user.getEmail());
49:                 userMap.put("role", user.getRole().name());
50:                 userMap.put("createdAt", user.getCreatedAt());
51:                 return userMap;
52:             })
53:             .collect(Collectors.toList());
54:
55:         Map<String, Object> response = new HashMap<>();
56:         response.put("items", items);
57:         response.put("claims", claims);
58:         response.put("users", userList);
59:         response.put("feedback", feedback);
60:
61:         return ResponseEntity.ok(response);
62:     }
63:
64:     @DeleteMapping("/items/{itemId}")
65:     public ResponseEntity<ApiResponse> deleteItem(
66:         @PathVariable Long itemId,
67:         @AuthenticationPrincipal UserPrincipal currentUser) {
68:
69:         itemService.deleteItem(itemId, currentUser);
70:
71:         ApiResponse response = ApiResponse.builder()
72:             .success(true)
73:             .message("Item deleted successfully")
74:             .build();
75:
76:         return ResponseEntity.ok(response);
77:     }
78:
79:     @DeleteMapping("/claims/{claimId}")
80:     public ResponseEntity<ApiResponse> deleteClaim(@PathVariable Long claimId) {
81:         claimService.deleteClaim(claimId);
82:
83:         ApiResponse response = ApiResponse.builder()
84:             .success(true)
85:             .message("Claim deleted successfully")
86:             .build();
87:
88:         return ResponseEntity.ok(response);
89:     }
90:
91:     @DeleteMapping("/users/{userId}")
92:     public ResponseEntity<ApiResponse> deleteUser(@PathVariable Long userId) {
93:         User user = userRepository.findById(userId)
94:             .orElseThrow(() -> new ResourceNotFoundException("User", "id", userId));
95:
96:         if (user.getRole() == User.Role.ADMIN) {
97:             throw new ResourceNotFoundException("Cannot delete admin user");
98:         }
99:
100:         userRepository.delete(user);
101:
102:         ApiResponse response = ApiResponse.builder()
103:             .success(true)
104:             .message("User deleted successfully")
105:             .build();
106:
107:         return ResponseEntity.ok(response);
108:     }
109:
110:     @DeleteMapping("/feedback/{feedbackId}")
111:     public ResponseEntity<ApiResponse> deleteFeedback(@PathVariable Long feedbackId) {
112:         feedbackService.deleteFeedback(feedbackId);
113:
114:         ApiResponse response = ApiResponse.builder()
115:             .success(true)
116:             .message("Feedback deleted successfully")
117:             .build();
118:
119:         return ResponseEntity.ok(response);
120:     }
121: }

```

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

```
=====
1: package com.lostandfound.controller;
2:
3: import com.lostandfound.dto.response.ApiResponse;
4: import com.lostandfound.security.UserPrincipal;
5: import com.lostandfound.service.ClaimService;
6: import lombok.RequiredArgsConstructor;
7: import org.springframework.http.ResponseEntity;
8: import org.springframework.security.core.annotation.AuthenticationPrincipal;
9: import org.springframework.web.bind.annotation.*;
10:
11: @RestController
12: @RequestMapping("/items")
13: @RequiredArgsConstructor
14: public class ClaimController {
15:
16:     private final ClaimService claimService;
17:
18:     @PostMapping("/{itemId}/claim")
19:     public ResponseEntity<ApiResponse> claimItem(
20:         @PathVariable Long itemId,
21:         @AuthenticationPrincipal UserPrincipal currentUser) {
22:
23:         ApiResponse response = claimService.claimItem(itemId, currentUser);
24:         return ResponseEntity.ok(response);
25:     }
26: }
```

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

```
=====
1: package com.lostandfound.controller;
2:
3: import com.lostandfound.dto.response.ClaimResponse;
4: import com.lostandfound.dto.response.DashboardResponse;
5: import com.lostandfound.dto.response.ItemResponse;
6: import com.lostandfound.dto.response.MessageResponse;
7: import com.lostandfound.security.UserPrincipal;
8: import com.lostandfound.service.ClaimService;
9: import com.lostandfound.service.ItemService;
10: import com.lostandfound.service.MessageService;
11: import lombok.RequiredArgsConstructor;
12: import org.springframework.http.ResponseEntity;
13: import org.springframework.security.core.annotation.AuthenticationPrincipal;
14: import org.springframework.web.bind.annotation.GetMapping;
15: import org.springframework.web.bind.annotation.RequestMapping;
16: import org.springframework.web.bind.annotation.RestController;
17:
18: import java.util.List;
19:
20: @RestController
21: @RequestMapping("/dashboard")
22: @RequiredArgsConstructor
23: public class DashboardController {
24:
25:     private final ItemService itemService;
26:     private final ClaimService claimService;
27:     private final MessageService messageService;
28:
29:     @GetMapping
30:     public ResponseEntity<DashboardResponse> getDashboard(
31:         @AuthenticationPrincipal UserPrincipal currentUser) {
32:
33:         List<ItemResponse> items = itemService.getUserItems(currentUser);
34:         List<ClaimResponse> claims = claimService.getUserClaims(currentUser);
35:         List<MessageResponse> messages = messageService.getUserMessages(currentUser);
36:
37:         String role = currentUser.getAuthorities().iterator().next()
38:             .getAuthority().replace("ROLE_", "");
39:
40:         DashboardResponse.UserInfo userInfo = DashboardResponse.UserInfo.builder()
41:             .name(currentUser.getName())
42:             .email(currentUser.getEmail())
43:             .role(role)
44:             .build();
45:
46:         DashboardResponse response = DashboardResponse.builder()
47:             .user(userInfo)
48:             .items(items)
49:             .claims(claims)
50:             .messages(messages)
51:             .build();
52:     }
```

```

53:         return ResponseEntity.ok(response);
54:     }
55: }

```

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

```

=====
1: package com.lostandfound.controller;
2:
3: import com.lostandfound.dto.request.FeedbackRequest;
4: import com.lostandfound.dto.response.ApiResponse;
5: import com.lostandfound.security.UserPrincipal;
6: import com.lostandfound.service.FeedbackService;
7: import jakarta.validation.Valid;
8: import lombok.RequiredArgsConstructor;
9: import org.springframework.http.ResponseEntity;
10: import org.springframework.security.core.annotation.AuthenticationPrincipal;
11: import org.springframework.web.bind.annotation.*;
12:
13: @RestController
14: @RequestMapping("/feedback")
15: @RequiredArgsConstructor
16: public class FeedbackController {
17:
18:     private final FeedbackService feedbackService;
19:
20:     @PostMapping
21:     public ResponseEntity<ApiResponse> submitFeedback(
22:         @Valid @RequestBody FeedbackRequest request,
23:         @AuthenticationPrincipal UserPrincipal currentUser) {
24:
25:         ApiResponse response = feedbackService.submitFeedback(request, currentUser);
26:         return ResponseEntity.ok(response);
27:     }
28: }

```

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

```

=====
1: package com.lostandfound.controller;
2:
3: import com.lostandfound.dto.request.ItemRequest;
4: import com.lostandfound.dto.response.ApiResponse;
5: import com.lostandfound.dto.response.ItemResponse;
6: import com.lostandfound.security.UserPrincipal;
7: import com.lostandfound.service.ItemService;
8: import jakarta.validation.Valid;
9: import lombok.RequiredArgsConstructor;
10: import org.springframework.http.ResponseEntity;
11: import org.springframework.security.core.annotation.AuthenticationPrincipal;
12: import org.springframework.web.bind.annotation.*;
13: import org.springframework.web.multipart.MultipartFile;
14:
15: import java.util.HashMap;
16: import java.util.List;
17: import java.util.Map;
18:
19: @RestController
20: @RequestMapping("/items")
21: @RequiredArgsConstructor
22: public class ItemController {
23:
24:     private final ItemService itemService;
25:
26:     @PostMapping
27:     public ResponseEntity<ApiResponse> createItem(
28:         @Valid @ModelAttribute ItemRequest request,
29:         @RequestParam(value = "image", required = false) MultipartFile image,
30:         @AuthenticationPrincipal UserPrincipal currentUser) {
31:
32:         ItemResponse itemResponse = itemService.createItem(request, image, currentUser);
33:
34:         ApiResponse response = ApiResponse.builder()
35:             .success(true)
36:             .message("Item registered successfully")
37:             .data(itemResponse)
38:             .build();
39:
40:         return ResponseEntity.ok(response);
41:     }
42:
43:     @GetMapping
44:     public ResponseEntity<Map<String, List<ItemResponse>>> getItems(

```

```

45:         @RequestParam(required = false, defaultValue = "") String search,
46:         @RequestParam(required = false, defaultValue = "") String status) {
47:
48:     List<ItemResponse> items = itemService.searchItems(search, status);
49:
50:     Map<String, List<ItemResponse>> response = new HashMap<>();
51:     response.put("items", items);
52:
53:     return ResponseEntity.ok(response);
54: }
55:
56: @DeleteMapping("/{itemId}")
57: public ResponseEntity<ApiResponse> deleteItem(
58:     @PathVariable Long itemId,
59:     @AuthenticationPrincipal UserPrincipal currentUser) {
60:
61:     itemService.deleteItem(itemId, currentUser);
62:
63:     ApiResponse response = ApiResponse.builder()
64:         .success(true)
65:         .message("Item deleted successfully")
66:         .build();
67:
68:     return ResponseEntity.ok(response);
69: }
70: }

```

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

```

1: package com.lostandfound.controller;
2:
3: import com.lostandfound.dto.request.MessageRequest;
4: import com.lostandfound.dto.response.ApiResponse;
5: import com.lostandfound.security.UserPrincipal;
6: import com.lostandfound.service.MessageService;
7: import jakarta.validation.Valid;
8: import lombok.RequiredArgsConstructor;
9: import org.springframework.http.ResponseEntity;
10: import org.springframework.security.core.annotation.AuthenticationPrincipal;
11: import org.springframework.web.bind.annotation.*;
12:
13: @RestController
14: @RequestMapping("/messages")
15: @RequiredArgsConstructor
16: public class MessageController {
17:
18:     private final MessageService messageService;
19:
20:     @PostMapping
21:     public ResponseEntity<ApiResponse> sendMessage(
22:         @Valid @RequestBody MessageRequest request,
23:         @AuthenticationPrincipal UserPrincipal currentUser) {
24:
25:         ApiResponse response = messageService.sendMessage(request, currentUser);
26:         return ResponseEntity.ok(response);
27:     }
28:
29:     @PostMapping("/reply")
30:     public ResponseEntity<ApiResponse> replyMessage(
31:         @Valid @RequestBody MessageRequest request,
32:         @AuthenticationPrincipal UserPrincipal currentUser) {
33:
34:         ApiResponse response = messageService.sendMessage(request, currentUser);
35:         return ResponseEntity.ok(response);
36:     }
37: }

```

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

```

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

```



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

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

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

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

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

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

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

```
=====
1: package com.lostandfound.dto.request;
2:
3: import jakarta.validation.constraints.Email;
4: import jakarta.validation.constraints.NotBlank;
5: import jakarta.validation.constraints.Pattern;
6: import jakarta.validation.constraints.Size;
7: import lombok.Data;
8: import lombok.AllArgsConstructor;
9: import lombok.NoArgsConstructor;
10:
11: @Data
```

```

12: @NoArgsConstructor
13: @AllArgsConstructor
14: public class RegisterRequest {
15:
16:     @NotBlank(message = "Name is required")
17:     @Size(min = 2, max = 100, message = "Name must be between 2 and 100 characters")
18:     @Pattern(regexp = "[a-zA-Z\\s]+$", message = "Name should only contain letters and sp
| aces")
19:     private String name;
20:
21:     @NotBlank(message = "Email is required")
22:     @Email(message = "Please provide a valid email address")
23:     @Size(max = 255, message = "Email must not exceed 255 characters")
24:     private String email;
25:
26:     @NotBlank(message = "Password is required")
27:     @Size(min = 8, max = 128, message = "Password must be between 8 and 128 characters")
28:     @Pattern(
29:         regexp = "^(?=.*[a-z])(?=.*[A-Z])(?=.*\\d)(?=.*[@$!%*?&])[A-Za-z\\d@$!%*?&]{8,
| }$",
30:         message = "Password must contain at least one uppercase letter, one lowercase
| letter, one number, and one special character"
31:     )
32:     private String password;
33:
34:     @NotBlank(message = "Confirm password is required")
35:     private String confirmPassword;
36: }

```

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

```

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

```

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

```

1: package com.lostandfound.dto.response;
2:
3: import lombok.AllArgsConstructor;
4: import lombok.Builder;
5: import lombok.Data;
6: import lombok.NoArgsConstructor;
7:
8: @Data
9: @Builder
10: @NoArgsConstructor
11: @AllArgsConstructor
12: public class AuthResponse {
13:     private boolean success;
14:     private String message;
15:     private String token;
16:     private UserDTO user;
17:
18:     @Data
19:     @Builder
20:     @NoArgsConstructor
21:     @AllArgsConstructor
22:     public static class UserDTO {
23:         private Long id;
24:         private String name;
25:         private String email;
26:         private String role;
27:     }
28: }

```

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

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

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

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

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

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

```
20: }
```

---

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

---

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

---

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

---

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

---

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

---

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

---

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

---

```
1: package com.lostandfound.exception;
2:
3: import com.lostandfound.dto.response.ApiResponse;
4: import org.slf4j.Logger;
5: import org.slf4j.LoggerFactory;
```

```

6: import org.springframework.http.HttpStatus;
7: import org.springframework.http.ResponseEntity;
8: import org.springframework.security.authentication.BadCredentialsException;
9: import org.springframework.security.core.userdetails.UsernameNotFoundException;
10: import org.springframework.validation.FieldError;
11: import org.springframework.web.bind.MethodArgumentNotValidException;
12: import org.springframework.web.bind.annotation.ExceptionHandler;
13: import org.springframework.web.bind.annotation.RestControllerAdvice;
14: import org.springframework.web.context.request.WebRequest;
15: import org.springframework.web.multipart.MaxUploadSizeExceededException;
16:
17: import java.util.HashMap;
18: import java.util.Map;
19:
20: @RestControllerAdvice
21: public class GlobalExceptionHandler {
22:
23:     private static final Logger logger = LoggerFactory.getLogger(GlobalExceptionHandler.cl
| ass);
24:
25:     @ExceptionHandler(ResourceNotFoundException.class)
26:     public ResponseEntity<ApiResponse> handleResourceNotFoundException(
27:         ResourceNotFoundException ex, WebRequest request) {
28:
29:         logger.warn("Resource not found: {}", ex.getMessage());
30:
31:         ApiResponse response = ApiResponse.builder()
32:             .success(false)
33:             .message(ex.getMessage())
34:             .build();
35:         return new ResponseEntity<>(response, HttpStatus.NOT_FOUND);
36:     }
37:
38:     @ExceptionHandler(BadRequestException.class)
39:     public ResponseEntity<ApiResponse> handleBadRequestException(
40:         BadRequestException ex, WebRequest request) {
41:
42:         logger.warn("Bad request: {}", ex.getMessage());
43:
44:         ApiResponse response = ApiResponse.builder()
45:             .success(false)
46:             .message(ex.getMessage())
47:             .build();
48:         return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
49:     }
50:
51:     @ExceptionHandler(UnauthorizedException.class)
52:     public ResponseEntity<ApiResponse> handleUnauthorizedException(
53:         UnauthorizedException ex, WebRequest request) {
54:
55:         logger.warn("Unauthorized access: {}", ex.getMessage());
56:
57:         ApiResponse response = ApiResponse.builder()
58:             .success(false)
59:             .message(ex.getMessage())
60:             .build();
61:         return new ResponseEntity<>(response, HttpStatus.UNAUTHORIZED);
62:     }
63:
64:     @ExceptionHandler(BadCredentialsException.class)
65:     public ResponseEntity<ApiResponse> handleBadCredentialsException(
66:         BadCredentialsException ex, WebRequest request) {
67:
68:         logger.warn("Invalid credentials attempt");
69:
70:         ApiResponse response = ApiResponse.builder()
71:             .success(false)
72:             .message("Invalid email or password")
73:             .build();
74:         return new ResponseEntity<>(response, HttpStatus.UNAUTHORIZED);
75:     }
76:
77:     @ExceptionHandler(UsernameNotFoundException.class)
78:     public ResponseEntity<ApiResponse> handleUsernameNotFoundException(
79:         UsernameNotFoundException ex, WebRequest request) {
80:
81:         logger.warn("User not found");
82:
83:         // Don't reveal if user exists or not for security
84:         ApiResponse response = ApiResponse.builder()
85:             .success(false)
86:             .message("Invalid email or password")
87:             .build();
88:         return new ResponseEntity<>(response, HttpStatus.UNAUTHORIZED);
89:     }

```

```

90:
91:     @ExceptionHandler(MethodArgumentNotValidException.class)
92:     public ResponseEntity<Map<String, Object>> handleValidationExceptions(
93:         MethodArgumentNotValidException ex) {
94:
95:         Map<String, String> errors = new HashMap<>();
96:         ex.getBindingResult().getAllErrors().forEach((error) -> {
97:             String fieldName = ((FieldError) error).getField();
98:             String errorMessage = error.getDefaultMessage();
99:             errors.put(fieldName, errorMessage);
100:         });
101:
102:         logger.warn("Validation errors: {}", errors);
103:
104:         Map<String, Object> response = new HashMap<>();
105:         response.put("success", false);
106:         response.put("message", "Validation failed");
107:         response.put("errors", errors);
108:
109:         return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
110:     }
111:
112:     @ExceptionHandler(MaxUploadSizeExceededException.class)
113:     public ResponseEntity<ApiResponse> handleMaxUploadSizeExceededException(
114:         MaxUploadSizeExceededException ex) {
115:
116:         logger.warn("File size exceeded");
117:
118:         ApiResponse response = ApiResponse.builder()
119:             .success(false)
120:             .message("File size exceeds maximum allowed size of 5MB")
121:             .build();
122:         return new ResponseEntity<>(response, HttpStatus.BAD_REQUEST);
123:     }
124:
125:     @ExceptionHandler(Exception.class)
126:     public ResponseEntity<ApiResponse> handleGlobalException(
127:         Exception ex, WebRequest request) {
128:
129:         // Log the full exception for debugging
130:         logger.error("Unexpected error occurred", ex);
131:
132:         // Don't expose internal error details to client
133:         ApiResponse response = ApiResponse.builder()
134:             .success(false)
135:             .message("An unexpected error occurred. Please try again later.")
136:             .build();
137:         return new ResponseEntity<>(response, HttpStatus.INTERNAL_SERVER_ERROR);
138:     }
139: }

```

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

```

1: package com.lostandfound.exception;
2:
3: import org.springframework.http.HttpStatus;
4: import org.springframework.web.bind.annotation.ResponseStatus;
5:
6: @ResponseStatus(HttpStatus.NOT_FOUND)
7: public class ResourceNotFoundException extends RuntimeException {
8:
9:     public ResourceNotFoundException(String message) {
10:         super(message);
11:     }
12:
13:     public ResourceNotFoundException(String resourceName, String fieldName, Object fieldValue) {
14:         super(String.format("%s not found with %s: '%s'", resourceName, fieldName, fieldValue));
15:     }
16: }

```

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

```

1: package com.lostandfound.exception;
2:
3: import org.springframework.http.HttpStatus;
4: import org.springframework.web.bind.annotation.ResponseStatus;
5:
6: @ResponseStatus(HttpStatus.UNAUTHORIZED)
7: public class UnauthorizedException extends RuntimeException {

```

```

8:
9:     public UnauthorizedException(String message) {
10:         super(message);
11:     }
12: }

```

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

```

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

```

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

```

=====
1: package com.lostandfound.model;
2:
3: import java.time.LocalDateTime;
4:
5: import org.hibernate.annotations.CreationTimestamp;
6:
7: import jakarta.persistence.Column;
8: import jakarta.persistence.Entity;
9: import jakarta.persistence.FetchType;
10: import jakarta.persistence.GeneratedValue;
11: import jakarta.persistence.GenerationType;
12: import jakarta.persistence.Id;
13: import jakarta.persistence.JoinColumn;
14: import jakarta.persistence.ManyToOne;
15: import jakarta.persistence.Table;
16: import lombok.AllArgsConstructor;
17: import lombok.Data;
18: import lombok.NoArgsConstructor;
19:
20: @Entity
21: @Table(name = "feedback")
22: @Data

```

```

23: @NoArgsConstructor
24: @AllArgsConstructor
25: public class Feedback {
26:
27:     @Id
28:     @GeneratedValue(strategy = GenerationType.IDENTITY)
29:     private Long id;
30:
31:     @ManyToOne(fetch = FetchType.LAZY)
32:     @JoinColumn(name = "user_id", nullable = false)
33:     private User user;
34:
35:     @Column(name = "feedback_text", nullable = false, columnDefinition = "TEXT")
36:     private String feedbackText;
37:
38:     @CreationTimestamp
39:     @Column(name = "submitted_at", nullable = false, updatable = false)
40:     private LocalDateTime submittedAt;
41: }

```

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

```

=====
1: package com.lostandfound.model;
2:
3: import java.time.LocalDateTime;
4:
5: import org.hibernate.annotations.CreationTimestamp;
6:
7: import jakarta.persistence.Column;
8: import jakarta.persistence.Entity;
9: import jakarta.persistence.EnumType;
10: import jakarta.persistence.Enumerated;
11: import jakarta.persistence.FetchType;
12: import jakarta.persistence.GeneratedValue;
13: import jakarta.persistence.GenerationType;
14: import jakarta.persistence.Id;
15: import jakarta.persistence.JoinColumn;
16: import jakarta.persistence.ManyToOne;
17: import jakarta.persistence.Table;
18: import lombok.AllArgsConstructor;
19: import lombok.Data;
20: import lombok.NoArgsConstructor;
21:
22: @Entity
23: @Table(name = "items")
24: @Data
25: @NoArgsConstructor
26: @AllArgsConstructor
27: public class Item {
28:
29:     @Id
30:     @GeneratedValue(strategy = GenerationType.IDENTITY)
31:     private Long id;
32:
33:     @Column(nullable = false)
34:     private String name;
35:
36:     @Column(nullable = false, columnDefinition = "TEXT")
37:     private String description;
38:
39:     @Column(nullable = false)
40:     private String location;
41:
42:     @Enumerated(EnumType.STRING)
43:     @Column(nullable = false)
44:     private Status status = Status.LOST;
45:
46:     @Column(name = "image")
47:     private String image;
48:
49:     @ManyToOne(fetch = FetchType.LAZY)
50:     @JoinColumn(name = "created_by", nullable = false)
51:     private User createdBy;
52:
53:     @CreationTimestamp
54:     @Column(name = "created_at", nullable = false, updatable = false)
55:     private LocalDateTime createdAt;
56:
57:     public enum Status {
58:         LOST, FOUND, CLAIMED
59:     }
60: }

```



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

```
=====
1: package com.lostandfound.model;
2:
3: import java.time.LocalDateTime;
4:
5: import org.hibernate.annotations.CreationTimestamp;
6:
7: import jakarta.persistence.Column;
8: import jakarta.persistence.Entity;
9: import jakarta.persistence.FetchType;
10: import jakarta.persistence.GeneratedValue;
11: import jakarta.persistence.GenerationType;
12: import jakarta.persistence.Id;
13: import jakarta.persistence.JoinColumn;
14: import jakarta.persistence.ManyToOne;
15: import jakarta.persistence.Table;
16: import lombok.AllArgsConstructor;
17: import lombok.Data;
18: import lombok.NoArgsConstructor;
19:
20: @Entity
21: @Table(name = "messages")
22: @Data
23: @NoArgsConstructor
24: @AllArgsConstructor
25: public class Message {
26:
27:     @Id
28:     @GeneratedValue(strategy = GenerationType.IDENTITY)
29:     private Long id;
30:
31:     @ManyToOne(fetch = FetchType.LAZY)
32:     @JoinColumn(name = "sender_id", nullable = false)
33:     private User sender;
34:
35:     @ManyToOne(fetch = FetchType.LAZY)
36:     @JoinColumn(name = "receiver_id", nullable = false)
37:     private User receiver;
38:
39:     @ManyToOne(fetch = FetchType.LAZY)
40:     @JoinColumn(name = "item_id", nullable = false)
41:     private Item item;
42:
43:     @Column(nullable = false, columnDefinition = "TEXT")
44:     private String message;
45:
46:     @CreationTimestamp
47:     @Column(name = "sent_at", nullable = false, updatable = false)
48:     private LocalDateTime sentAt;
49: }
```

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

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

```

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

```

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

```

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

```

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

```

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

```

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

```

1: package com.lostandfound.repository;
2:
3: import com.lostandfound.model.Item;
4: import com.lostandfound.model.Item.Status;
5: import com.lostandfound.model.User;

```

```

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

```

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

```

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

```

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

```

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

```

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

```

1: package com.lostandfound.scheduler;
2:
3: import com.lostandfound.model.Claim;
4: import com.lostandfound.model.Item;
5: import com.lostandfound.repository.ClaimRepository;
6: import com.lostandfound.repository.ItemRepository;
7: import lombok.RequiredArgsConstructor;
8: import org.slf4j.Logger;
9: import org.slf4j.LoggerFactory;
10: import org.springframework.scheduling.annotation.Scheduled;
11: import org.springframework.stereotype.Component;
12: import org.springframework.transaction.annotation.Transactional;
13:
14: import java.time.LocalDateTime;
15: import java.util.List;
16:

```

```

17: @Component
18: @RequiredArgsConstructor
19: public class ClaimCleanupScheduler {
20:
21:     private static final Logger logger = LoggerFactory.getLogger(ClaimCleanupScheduler.class);
22:
23:     private final ClaimRepository claimRepository;
24:     private final ItemRepository itemRepository;
25:
26:     @Scheduled(cron = "0 0 2 * * ?") // Run every day at 2 AM
27:     @Transactional
28:     public void cleanupOldClaims() {
29:         logger.info("Starting cleanup of old claims...");
30:
31:         try {
32:             LocalDateTime cutoffDate = LocalDateTime.now().minusDays(7);
33:             List<Claim> oldClaims = claimRepository.findOldClaims(cutoffDate);
34:
35:             int claimsDeleted = oldClaims.size();
36:
37:             for (Claim claim : oldClaims) {
38:                 Item item = claim.getItem();
39:                 claimRepository.delete(claim);
40:
41:                 // Update item status if it was claimed
42:                 if (item.getStatus() == Item.Status.CLAIMED) {
43:                     item.setStatus(Item.Status.FOUND);
44:                     itemRepository.save(item);
45:                 }
46:             }
47:
48:             logger.info("Deleted {} old claims and updated item statuses", claimsDeleted);
49:         } catch (Exception e) {
50:             logger.error("Error during claim cleanup", e);
51:         }
52:     }
53: }

```

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

```

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

```

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

```
=====
1: package com.lostandfound.security;
2:
3: import jakarta.servlet.FilterChain;
4: import jakarta.servlet.ServletException;
5: import jakarta.servlet.http.HttpServletRequest;
6: import jakarta.servlet.http.HttpServletResponse;
7: import lombok.RequiredArgsConstructor;
8: import org.slf4j.Logger;
9: import org.slf4j.LoggerFactory;
10: import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
11: import org.springframework.security.core.context.SecurityContextHolder;
12: import org.springframework.security.core.userdetails.UserDetails;
13: import org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
14: import org.springframework.stereotype.Component;
15: import org.springframework.util.StringUtils;
16: import org.springframework.web.filter.OncePerRequestFilter;
17:
18: import java.io.IOException;
19:
20: @Component
21: @RequiredArgsConstructor
22: public class JwtAuthenticationFilter extends OncePerRequestFilter {
23:
24:     private static final Logger logger = LoggerFactory.getLogger(JwtAuthenticationFilter.class);
25:
26:     private final JwtTokenProvider tokenProvider;
27:     private final CustomUserDetailsService customUserDetailsService;
28:
29:     @Override
30:     protected void doFilterInternal(HttpServletRequest request,
31:                                     HttpServletResponse response,
32:                                     FilterChain filterChain) throws ServletException, IOException {
33:         try {
34:             String jwt = getJwtFromRequest(request);
35:
36:             if (StringUtils.hasText(jwt) && tokenProvider.validateToken(jwt)) {
37:                 Long userId = tokenProvider.getUserIdFromToken(jwt);
38:                 UserDetails userDetails = customUserDetailsService.loadUserById(userId);
39:
40:                 UsernamePasswordAuthenticationToken authentication =
41:                     new UsernamePasswordAuthenticationToken(userDetails, null, userDetails.getAuthorities());
42:                 authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));
43:
44:                 SecurityContextHolder.getContext().setAuthentication(authentication);
45:             } catch (Exception ex) {
46:                 logger.error("Could not set user authentication in security context", ex);
47:             }
48:
49:             filterChain.doFilter(request, response);
50:         }
51:     }
52:
53:     private String getJwtFromRequest(HttpServletRequest request) {
54:         String bearerToken = request.getHeader("Authorization");
55:         if (StringUtils.hasText(bearerToken) && bearerToken.startsWith("Bearer ")) {
56:             return bearerToken.substring(7);
57:         }
58:         return null;
59:     }
60: }
```

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

```
=====
1: package com.lostandfound.security;
2:
3: import io.jsonwebtoken.*;
4: import io.jsonwebtoken.security.Keys;
5: import org.slf4j.Logger;
6: import org.slf4j.LoggerFactory;
7: import org.springframework.beans.factory.annotation.Value;
8: import org.springframework.security.core.Authentication;
9: import org.springframework.stereotype.Component;
10:
11: import javax.crypto.SecretKey;
12: import java.util.Date;
13:
14: @Component
```

```

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

```

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

```

1: package com.lostandfound.security;
2:
3: import com.lostandfound.model.User;
4: import lombok.AllArgsConstructor;
5: import lombok.Data;
6: import org.springframework.security.core.GrantedAuthority;
7: import org.springframework.security.core.authority.SimpleGrantedAuthority;
8: import org.springframework.security.core.userdetails.UserDetails;
9:
10: import java.util.Collection;
11: import java.util.Collections;
12:
13: @Data
14: @AllArgsConstructor
15: public class UserPrincipal implements UserDetails {
16:
17:     private Long id;
18:     private String name;
19:     private String email;
20:     private String password;
21:     private Collection<? extends GrantedAuthority> authorities;
22:
23:     public static UserPrincipal create(User user) {
24:         Collection<GrantedAuthority> authorities = Collections.singleton(
25:             new SimpleGrantedAuthority("ROLE_" + user.getRole().name())
26:         );

```

```

27:
28:         return new UserPrincipal(
29:             user.getId(),
30:             user.getName(),
31:             user.getEmail(),
32:             user.getPassword(),
33:             authorities
34:         );
35:     }
36:
37:     @Override
38:     public String getUsername() {
39:         return email;
40:     }
41:
42:     @Override
43:     public boolean isAccountNonExpired() {
44:         return true;
45:     }
46:
47:     @Override
48:     public boolean isAccountNonLocked() {
49:         return true;
50:     }
51:
52:     @Override
53:     public boolean isCredentialsNonExpired() {
54:         return true;
55:     }
56:
57:     @Override
58:     public boolean isEnabled() {
59:         return true;
60:     }
61: }

```

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

```

=====
1: package com.lostandfound.service;
2:
3: import com.lostandfound.dto.response.ApiResponse;
4: import com.lostandfound.dto.response.ClaimResponse;
5: import com.lostandfound.exception.BadRequestException;
6: import com.lostandfound.exception.ResourceNotFoundException;
7: import com.lostandfound.model.Claim;
8: import com.lostandfound.model.Item;
9: import com.lostandfound.model.User;
10: import com.lostandfound.repository.ClaimRepository;
11: import com.lostandfound.repository.ItemRepository;
12: import com.lostandfound.repository.UserRepository;
13: import com.lostandfound.security.UserPrincipal;
14: import lombok.RequiredArgsConstructor;
15: import org.springframework.stereotype.Service;
16: import org.springframework.transaction.annotation.Transactional;
17:
18: import java.util.List;
19: import java.util.stream.Collectors;
20:
21: @Service
22: @RequiredArgsConstructor
23: public class ClaimService {
24:
25:     private final ClaimRepository claimRepository;
26:     private final ItemRepository itemRepository;
27:     private final UserRepository userRepository;
28:
29:     @Transactional
30:     public ApiResponse claimItem(Long itemId, UserPrincipal currentUser) {
31:         Item item = itemRepository.findById(itemId)
32:             .orElseThrow(() -> new ResourceNotFoundException("Item", "id", itemId));
33:
34:         User user = userRepository.findById(currentUser.getId())
35:             .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
36: | .getId()));
37:
38:         if (item.getStatus() == Item.Status.CLAIMED) {
39:             throw new BadRequestException("Item already claimed");
40:         }
41:
42:         if (claimRepository.existsByItemAndClaimedBy(item, user)) {
43:             throw new BadRequestException("You have already claimed this item");
44:         }

```

```

45:         Claim claim = new Claim();
46:         claim.setItem(item);
47:         claim.setClaimedBy(user);
48:         claim.setClaimantName(user.getName());
49:         claim.setClaimantEmail(user.getEmail());
50:
51:         claimRepository.save(claim);
52:
53:         // Update item status
54:         item.setStatus(Item.Status.CLAIMED);
55:         itemRepository.save(item);
56:
57:         return ApiResponse.builder()
58:             .success(true)
59:             .message("Item claimed successfully")
60:             .build();
61:     }
62:
63:     @Transactional(readOnly = true)
64:     public List<ClaimResponse> getUserClaims(UserPrincipal currentUser) {
65:         User user = userRepository.findById(currentUser.getId())
66:             .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
| .getId()));
67:
68:         List<Claim> claims = claimRepository.findByClaimedByOrderByClaimedAtDesc(user);
69:         return claims.stream()
70:             .map(this::mapToClaimResponse)
71:             .collect(Collectors.toList());
72:     }
73:
74:     @Transactional
75:     public void deleteClaim(Long claimId) {
76:         Claim claim = claimRepository.findById(claimId)
77:             .orElseThrow(() -> new ResourceNotFoundException("Claim", "id", claimId));
78:
79:         Item item = claim.getItem();
80:
81:         claimRepository.delete(claim);
82:
83:         // Update item status back to FOUND if it was CLAIMED
84:         if (item.getStatus() == Item.Status.CLAIMED) {
85:             item.setStatus(Item.Status.FOUND);
86:             itemRepository.save(item);
87:         }
88:     }
89:
90:     private ClaimResponse mapToClaimResponse(Claim claim) {
91:         return ClaimResponse.builder()
92:             .id(claim.getId())
93:             .itemId(claim.getItem().getId())
94:             .itemName(claim.getItem().getName())
95:             .description(claim.getItem().getDescription())
96:             .location(claim.getItem().getLocation())
97:             .claimedBy(claim.getClaimedBy().getId())
98:             .claimerName(claim.getClaimedBy().getName())
99:             .claimantName(claim.getClaimantName())
100:             .claimantEmail(claim.getClaimantEmail())
101:             .claimedAt(claim.getClaimedAt())
102:             .build();
103:     }
104: }

```

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

```

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

```



```

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

```

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

```

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

```

```

29:         if (originalFileName.contains(".")) {
30:             throw new BadRequestException("Filename contains invalid path sequence " +
| originalFileName);
31:         }
32:
33:         // Create unique filename
34:         String fileExtension = "";
35:         if (originalFileName.contains(".")) {
36:             fileExtension = originalFileName.substring(originalFileName.lastIndexOf(".")
| "));
37:         }
38:         String uniqueFileName = UUID.randomUUID().toString() + "_" + originalFileName;
39:
40:         // Create directory if it doesn't exist
41:         Path fileStorageLocation = Paths.get(fileStorageProperties.getUploadDir())
42:             .toAbsolutePath().normalize();
43:         Files.createDirectories(fileStorageLocation);
44:
45:         // Copy file to the target location
46:         Path targetLocation = fileStorageLocation.resolve(uniqueFileName);
47:         Files.copy(file.getInputStream(), targetLocation, StandardCopyOption.REPLACE_E
| XISTING);
48:
49:         return "uploads/" + uniqueFileName;
50:     } catch (IOException ex) {
51:         throw new BadRequestException("Could not store file " + originalFileName + ".
| Please try again!");
52:     }
53: }
54:
55: public void deleteFile(String filePath) {
56:     try {
57:         if (filePath != null && filePath.startsWith("uploads/")) {
58:             String fileName = filePath.replace("uploads/", "");
59:             Path fileStorageLocation = Paths.get(fileStorageProperties.getUploadDir())
60:                 .toAbsolutePath().normalize();
61:             Path targetLocation = fileStorageLocation.resolve(fileName);
62:             Files.deleteIfExists(targetLocation);
63:         }
64:     } catch (IOException ex) {
65:         // Log error but don't throw exception
66:         System.err.println("Could not delete file: " + filePath);
67:     }
68: }
69: }

```

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

```

=====
1: package com.lostandfound.service;
2:
3: import com.lostandfound.dto.request.ItemRequest;
4: import com.lostandfound.dto.response.ItemResponse;
5: import com.lostandfound.exception.BadRequestException;
6: import com.lostandfound.exception.ResourceNotFoundException;
7: import com.lostandfound.model.Item;
8: import com.lostandfound.model.User;
9: import com.lostandfound.repository.ClaimRepository;
10: import com.lostandfound.repository.ItemRepository;
11: import com.lostandfound.repository.MessageRepository;
12: import com.lostandfound.repository.UserRepository;
13: import com.lostandfound.security.UserPrincipal;
14: import lombok.RequiredArgsConstructor;
15: import org.springframework.stereotype.Service;
16: import org.springframework.transaction.annotation.Transactional;
17: import org.springframework.web.multipart.MultipartFile;
18:
19: import java.util.List;
20: import java.util.stream.Collectors;
21:
22: @Service
23: @RequiredArgsConstructor
24: public class ItemService {
25:
26:     private final ItemRepository itemRepository;
27:     private final UserRepository userRepository;
28:     private final ClaimRepository claimRepository;
29:     private final MessageRepository messageRepository;
30:     private final FileStorageService fileStorageService;
31:
32:     @Transactional
33:     public ItemResponse createItem(ItemRequest request, MultipartFile image, UserPrincipal
| currentUser) {
34:         User user = userRepository.findById(currentUser.getId())

```

```

35:         .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
| .getId()));
36:
37:         String imagePath = null;
38:         if (image != null && !image.isEmpty()) {
39:             imagePath = fileStorageService.storeFile(image);
40:         }
41:
42:         Item item = new Item();
43:         item.setName(request.getName());
44:         item.setDescription(request.getDescription());
45:         item.setLocation(request.getLocation());
46:         item.setStatus(Item.Status.valueOf(request.getStatus().toUpperCase()));
47:         item.setImage(imagePath);
48:         item.setCreatedBy(user);
49:
50:         item = itemRepository.save(item);
51:
52:         return mapToItemResponse(item);
53:     }
54:
55:     @Transactional(readOnly = true)
56:     public List<ItemResponse> searchItems(String search, String status) {
57:         Item.Status itemStatus = null;
58:         if (status != null && !status.isEmpty() && !status.equals("all")) {
59:             try {
60:                 itemStatus = Item.Status.valueOf(status.toUpperCase());
61:             } catch (IllegalArgumentException e) {
62:                 throw new BadRequestException("Invalid status: " + status);
63:             }
64:         }
65:
66:         List<Item> items = itemRepository.searchItems(search, itemStatus);
67:         return items.stream()
68:             .map(this::mapToItemResponse)
69:             .collect(Collectors.toList());
70:     }
71:
72:     @Transactional(readOnly = true)
73:     public List<ItemResponse> getUserItems(UserPrincipal currentUser) {
74:         User user = userRepository.findById(currentUser.getId())
75:         .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
| .getId()));
76:
77:         List<Item> items = itemRepository.findByCreatedByOrderByCreatedAtDesc(user);
78:         return items.stream()
79:             .map(this::mapToItemResponse)
80:             .collect(Collectors.toList());
81:     }
82:
83:     @Transactional
84:     public void deleteItem(Long itemId, UserPrincipal currentUser) {
85:         Item item = itemRepository.findById(itemId)
86:         .orElseThrow(() -> new ResourceNotFoundException("Item", "id", itemId));
87:
88:         // Check if user is admin or item owner
89:         User user = userRepository.findById(currentUser.getId())
90:         .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
| .getId()));
91:
92:         if (!user.getRole().equals(User.Role.ADMIN) && !item.getCreatedBy().getId().equals
| (currentUser.getId())) {
93:             throw new BadRequestException("You don't have permission to delete this item")
| ;
94:         }
95:
96:         // Delete related claims and messages
97:         claimRepository.deleteAll(claimRepository.findByItem(item));
98:         messageRepository.deleteAll(messageRepository.findAll().stream()
99:             .filter(m -> m.getItem().getId().equals(itemId))
100:             .collect(Collectors.toList()));
101:
102:         // Delete image file if exists
103:         if (item.getImage() != null) {
104:             fileStorageService.deleteFile(item.getImage());
105:         }
106:
107:         itemRepository.delete(item);
108:     }
109:
110:     private ItemResponse mapToItemResponse(Item item) {
111:         return ItemResponse.builder()
112:             .id(item.getId())
113:             .name(item.getName())
114:             .description(item.getDescription())

```

```

115:         .location(item.getLocation())
116:         .status(item.getStatus().name())
117:         .image(item.getImage())
118:         .createdBy(item.getCreatedBy().getId())
119:         .creatorName(item.getCreatedBy().getName())
120:         .createdAt(item.getCreatedAt())
121:         .build();
122:     }
123: }

```

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

```

=====
1: package com.lostandfound.service;
2:
3: import com.lostandfound.dto.request.MessageRequest;
4: import com.lostandfound.dto.response.ApiResponse;
5: import com.lostandfound.dto.response.MessageResponse;
6: import com.lostandfound.exception.ResourceNotFoundException;
7: import com.lostandfound.model.Item;
8: import com.lostandfound.model.Message;
9: import com.lostandfound.model.User;
10: import com.lostandfound.repository.ItemRepository;
11: import com.lostandfound.repository.MessageRepository;
12: import com.lostandfound.repository.UserRepository;
13: import com.lostandfound.security.UserPrincipal;
14: import lombok.RequiredArgsConstructor;
15: import org.springframework.stereotype.Service;
16: import org.springframework.transaction.annotation.Transactional;
17:
18: import java.util.List;
19: import java.util.stream.Collectors;
20:
21: @Service
22: @RequiredArgsConstructor
23: public class MessageService {
24:
25:     private final MessageRepository messageRepository;
26:     private final UserRepository userRepository;
27:     private final ItemRepository itemRepository;
28:
29:     @Transactional
30:     public ApiResponse sendMessage(MessageRequest request, UserPrincipal currentUser) {
31:         User sender = userRepository.findById(currentUser.getId())
32:             .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
| .getId()));
33:
34:         User receiver = userRepository.findById(request.getReceiverId())
35:             .orElseThrow(() -> new ResourceNotFoundException("User", "id", request.get
| ReceiverId()));
36:
37:         Item item = itemRepository.findById(request.getItemId())
38:             .orElseThrow(() -> new ResourceNotFoundException("Item", "id", request.get
| ItemId()));
39:
40:         Message message = new Message();
41:         message.setSender(sender);
42:         message.setReceiver(receiver);
43:         message.setItem(item);
44:         message.setMessage(request.getMessage());
45:
46:         messageRepository.save(message);
47:
48:         return ApiResponse.builder()
49:             .success(true)
50:             .message("Message sent successfully")
51:             .build();
52:     }
53:
54:     @Transactional(readOnly = true)
55:     public List<MessageResponse> getUserMessages(UserPrincipal currentUser) {
56:         User user = userRepository.findById(currentUser.getId())
57:             .orElseThrow(() -> new ResourceNotFoundException("User", "id", currentUser
| .getId()));
58:
59:         List<Message> messages = messageRepository.findByReceiverOrderBySentAtDesc(user);
60:         return messages.stream()
61:             .map(this::mapToMessageResponse)
62:             .collect(Collectors.toList());
63:     }
64:
65:     private MessageResponse mapToMessageResponse(Message message) {
66:         return MessageResponse.builder()
67:             .id(message.getId())

```

```

68:         .senderId(message.getSender().getId())
69:         .senderName(message.getSender().getName())
70:         .receiverId(message.getReceiver().getId())
71:         .itemId(message.getItem().getId())
72:         .itemName(message.getItem().getName())
73:         .message(message.getMessage())
74:         .sentAt(message.getSentAt())
75:         .build();
76:     }
77: }

```

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

```

=====
1: package com.lostandfound.service;
2:
3: import com.lostandfound.dto.request.LoginRequest;
4: import com.lostandfound.dto.request.RegisterRequest;
5: import com.lostandfound.dto.response.AuthResponse;
6: import com.lostandfound.exception.BadRequestException;
7: import com.lostandfound.model.User;
8: import com.lostandfound.repository.UserRepository;
9: import com.lostandfound.security.JwtTokenProvider;
10: import com.lostandfound.security.UserPrincipal;
11: import lombok.RequiredArgsConstructor;
12: import org.slf4j.Logger;
13: import org.slf4j.LoggerFactory;
14: import org.springframework.security.authentication.AuthenticationManager;
15: import org.springframework.security.authentication.BadCredentialsException;
16: import org.springframework.security.authentication.LockedException;
17: import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
18: import org.springframework.security.core.Authentication;
19: import org.springframework.security.core.context.SecurityContextHolder;
20: import org.springframework.security.crypto.password.PasswordEncoder;
21: import org.springframework.stereotype.Service;
22: import org.springframework.transaction.annotation.Transactional;
23:
24: @Service
25: @RequiredArgsConstructor
26: public class UserService {
27:
28:     private static final Logger logger = LoggerFactory.getLogger(UserService.class);
29:
30:     private final UserRepository userRepository;
31:     private final PasswordEncoder passwordEncoder;
32:     private final AuthenticationManager authenticationManager;
33:     private final JwtTokenProvider tokenProvider;
34:
35:     @Transactional
36:     public AuthResponse registerUser(RegisterRequest request) {
37:         // Normalize email to lowercase
38:         String email = request.getEmail().toLowerCase().trim();
39:
40:         // Check if email already exists
41:         if (userRepository.existsByEmail(email)) {
42:             logger.warn("Registration attempt with existing email: {}", email);
43:             throw new BadRequestException("Email address is already registered");
44:         }
45:
46:         // Validate password confirmation
47:         if (!request.getPassword().equals(request.getConfirmPassword())) {
48:             throw new BadRequestException("Passwords do not match");
49:         }
50:
51:         // Create new user
52:         User user = new User();
53:         user.setName(request.getName().trim());
54:         user.setEmail(email);
55:         user.setPassword(passwordEncoder.encode(request.getPassword()));
56:         user.setRole(User.Role.USER);
57:
58:         try {
59:             user = userRepository.save(user);
60:             logger.info("New user registered successfully: {}", email);
61:         } catch (Exception e) {
62:             logger.error("Error during user registration: {}", email, e);
63:             throw new BadRequestException("Registration failed. Please try again.");
64:         }
65:
66:         // Auto-login after registration
67:         Authentication authentication = authenticationManager.authenticate(
68:             new UsernamePasswordAuthenticationToken(email, request.getPassword())
69:         );
70:
=====

```

```

71:         SecurityContextHolder.getContext().setAuthentication(authentication);
72:         String jwt = tokenProvider.generateToken(authentication);
73:
74:         // Build user DTO
75:         AuthResponse.UserDTO userDTO = AuthResponse.UserDTO.builder()
76:             .id(user.getId())
77:             .name(user.getName())
78:             .email(user.getEmail())
79:             .role(user.getRole().name())
80:             .build();
81:
82:         return AuthResponse.builder()
83:             .success(true)
84:             .message("Registration successful")
85:             .token(jwt)
86:             .user(userDTO)
87:             .build();
88:     }
89:
90:     @Transactional(readOnly = true)
91:     public AuthResponse loginUser(LoginRequest request) {
92:         String email = request.getEmail().toLowerCase().trim();
93:
94:         try {
95:             // Authenticate user
96:             Authentication authentication = authenticationManager.authenticate(
97:                 new UsernamePasswordAuthenticationToken(email, request.getPassword())
98:             );
99:
100:            SecurityContextHolder.getContext().setAuthentication(authentication);
101:
102:            // Generate JWT token
103:            String jwt = tokenProvider.generateToken(authentication);
104:
105:            // Fetch user details
106:            User user = userRepository.findById(email)
107:                .orElseThrow(() -> new BadRequestException("User not found"));
108:
109:            logger.info("User logged in successfully: {}", email);
110:
111:            // Build user DTO
112:            AuthResponse.UserDTO userDTO = AuthResponse.UserDTO.builder()
113:                .id(user.getId())
114:                .name(user.getName())
115:                .email(user.getEmail())
116:                .role(user.getRole().name())
117:                .build();
118:
119:            return AuthResponse.builder()
120:                .success(true)
121:                .message("Login successful")
122:                .token(jwt)
123:                .user(userDTO)
124:                .build();
125:
126:        } catch (BadCredentialsException e) {
127:            logger.warn("Failed login attempt for email: {}", email);
128:            throw new BadCredentialsException("Invalid email or password");
129:        } catch (LockedException e) {
130:            logger.warn("Login attempt for locked account: {}", email);
131:            throw new BadRequestException("Account is locked. Please contact support.");
132:        } catch (Exception e) {
133:            logger.error("Error during login for email: {}", email, e);
134:            throw new BadRequestException("Login failed. Please try again.");
135:        }
136:    }
137:
138:    @Transactional(readOnly = true)
139:    public User getUserById(Long userId) {
140:        return userRepository.findById(userId)
141:            .orElseThrow(() -> new BadRequestException("User not found"));
142:    }
143:
144:    @Transactional(readOnly = true)
145:    public User getUserByEmail(String email) {
146:        return userRepository.findById(email.toLowerCase().trim())
147:            .orElseThrow(() -> new BadRequestException("User not found"));
148:    }
149: }

```

---